



DSPACE

<https://dspace.org/>

A systems architecture for IoT connected-edge runtime configuration

Said, Ebrahim Y

2022

<https://hdl.handle.net/10539/34395>

A Systems Architecture for IoT Connected-edge Runtime Configuration



Ebrahim Y. Said
Faculty of Engineering & the Built Environment
University of the Witwatersrand

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Johannesburg 2022

Declaration

I declare that this thesis is my own, unaided work, other than where specifically acknowledged. It is being submitted for the degree of Doctor of Philosophy at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.

A handwritten signature in cursive script, reading "E.Y. Said".

Ebrahim Y. Said

May, 2022

Abstract

This thesis presents a systems architecture method that applies object, process, and state as primary attributes for modelling smart devices in an IoT application ecosystem. This systems architecture method is grounded in pure systems thinking that enables a significant focus on systems purpose using a state-of-the-art modelling notation while supporting complexity reduction in systems architecture model representation.

In current research, the Internet of Things (IoT) body of work mainly applies functional and domain coverage related to the representation of its architecture. This abstraction aims to hide enabling technologies while simultaneously trying to handle system complexity. Increases in complexity due to the evolution of IoT edge systems have seen requisite increases in technological needs advancement. IoT edge systems enable the physical extensions of the internet that exists today. This cyber-physical evolution goes beyond the services that individual embedded devices expose. It underpins the IoT by creating connected edge systems that integrate large-scale digital and physical processes. The variation in heterogeneity predominantly causes increases in complexity in these systems. A systems approach can reduce the complexity in stating a systems architecture for the IoT Connected-edge (IoT-Ce). This approach stems from the belief that low-level IoT-Ce subsystem characteristics that influence the state of the system can abstract to a conceptual IoT system architecture to increase the accuracy of model dependability through increased state awareness.

This thesis presents two contributions. Firstly, the study proposes a standards-based systems architecture method that outlines the systems, subsystems, and components employed at runtime in the IoT-Ce. The Object Process Methodology (OPM) ISO-19450 guides the modelling of the systems architecture. The **IoT-Ce Systems Architecture for Runtime Configuration (ISARC)** suggests a hierarchical structure for a multi-layered connected heterogeneous IoT edge system. The study of the runtime properties of systems can provide a better understanding of how to model systems that are dependable and scalable. Additionally, these systems have the essential properties to find application in large-scale

distributed systems architecture. The primary contribution is the suggestion that identifying well-defined system, subsystem, and component boundaries using a known approach supports complexity reduction in IoT-Ce models. The proposed approach provides the method for testing variations of configuration for the IoT-Ce.

Secondly, the thesis provides a method to validate the ISARC using OPM and Design Structure Matrices (DSM). The method for modelling configuration decisions for the IoT-Ce uses OPM as the basis for decision parameter extraction and DSM to represent the decision structure. A test of conceptual architecture variations of the ISARC uses a novel concrete control design based on instantaneous and accumulated errors. This contribution's primary benefit extends the systems architecture by proving that the ISARC can support ongoing runtime configuration decisions using a series of Python simulated data sets. Finally, these data sets compare a historical data set that mimics a hierarchical systems architecture to argue the applicability of parameter-based architecture decisions for the IoT-Ce.

The ISARC aims to position an approach to simplify system complexities by suggesting the function and form of the IoT-Ce. The effort to reduce complexity in design is in line with an existing body of research in model-based systems engineering using the OPM ISO-19450 standard.

Keywords IoT System Architecture, OPM ISO-19450, IoT Connected Edge, Runtime Configuration, Design Structure Matrix, Architecture Decisions, PI Control, Python

Acknowledgements

First and foremost, my eternal thanks to the Almighty for his blessings, will, and strength that he has bestowed on me. Without this, I would not have been set on the path to find purpose and drive in delivering this thesis.

I would like to thank my supervisor, Prof. Ken J. Nixon, for his knowledge, guidance, and steadfast support throughout the duration of this research. I would also like to thank Prof. Ivan Hofsajer for his guidance in setting me on the path to uncovering the meaning of research. Your words are appreciated and will steer me through my future endeavors. A special thanks go to my friends and colleagues at MTN South Africa for their support and advice. Your trust in my ability to walk both career and academic journeys has been part of my success.

I would like to thank my parents, Firdows and Sulaiman Said. Your patience and continued guidance have shown me that nothing is impossible, even in the face of personal challenges. Finally, I would like to thank my children, Sara and Kalil, and my wife, Natasha Said. In the words of Maya Angelou, “This is a wonderful day. I’ve never seen this one before.”

This work was performed with the Department of Electrical and Information Engineering, at the University of the Witwatersrand, Johannesburg.

Contents

Declaration	i
Abstract	ii
Acknowledgements	iv
1 The Evolution of Things	1
1.1 Introduction	1
1.2 Background	2
1.3 Overview of a Fragmented IoT Ecosystem	5
1.4 Dealing with the Edge	8
1.5 Thesis Objectives	10
1.6 Literature Overview	13
1.6.1 Systems Architecture	13
1.6.2 IoT Connected-edge	16
1.6.3 Runtime Configuration	18
1.6.4 Architecture Configuration Decisions	20
1.6.5 A Converged Literature Summary	22
1.7 Scope & Constraints	23
1.8 Approach & Limitations	31
1.9 Thesis Structure	39
1.10 Chapter Summary	40
2 An IoT Systems Architecture	43
2.1 An existing approach for supporting IoT Architectures	44
2.2 Extracting Depth in Vertical Themes	46
2.2.1 Systems Architecture	46
2.2.1.1 A Systems approach towards Architecture for IoT	46
2.2.1.2 About Structured Systems Design	50

2.2.1.3	About Models	55
2.2.1.4	About Model Simplification & Analysis	59
2.2.1.5	About State	63
2.2.2	IoT Connected-edge	68
2.2.2.1	Surfing the Mobile Wave	70
2.2.2.2	Classifying IoT Protocols	74
2.2.3	Runtime Configuration	79
2.2.3.1	Component Arrangement	86
2.2.3.2	System Configuration Capabilities	89
2.2.4	Architecture Configuration Decisions	96
2.3	Chapter Summary	101
3	Systems Architectures and IoT-Ce Configuration	103
3.1	ISARC Requirements Extraction	104
3.1.1	Additional Conformance Requirements	108
3.2	State of the Art Systems Architecture Realisation	111
3.2.1	Architecture Guidance on Form & Function	113
3.3	ISARC Vertical Theme Depth Convergence	116
3.4	System Description	123
3.4.1	System Form and Function	124
3.4.2	System Interfaces & States	126
3.4.3	Configuration Variation	130
3.5	Architecture Decisions and Evaluation Criteria	132
3.6	Chapter Summary	133
4	Modelling the ISARC: Systems, IoT-Ce and Configuration	136
4.1	System Purpose & Primary Processes	137
4.1.1	System Diagram (SD): IoT-Ce System Purpose	138
4.1.2	System Diagram (SD1): IoT Edge System Value Delivering	140
4.1.3	System Diagram (SD1.1): IoT Edge Data Transmitting	143
4.2	Supporting Objects & Processes	145
4.3	System Interfaces & States	153
4.4	Runtime Configuration Variation	158
4.4.1	Applied Process State Transition	161
4.5	ISARC Form & Function Summary	165
4.6	System Configuration Decision Model	168
4.6.1	Decision Model 1 Inputs	170

4.6.2	Decision Model 2 Inputs	171
4.6.3	Decision Model 3 Inputs	172
4.6.4	Decision Model 4 Inputs	173
4.7	Chapter Summary	174
5	Examples and Evidence: ISARC Model Validations	176
5.1	Primary SD, SDx, SDx.x Model Overview	177
5.2	SDx.x.x Model Extensions	180
5.2.1	Device Identifying	180
5.2.2	Device Discovering	182
5.2.3	Device Managing	189
5.3	Model Validation Evidence	193
5.4	SDn Model Extension Summary	196
5.5	SDn Decision Model Validation	197
5.5.1	Model Validation Implementation	197
5.5.2	ISARC DSM Decision Model 1 to 4 Initialisation	198
5.5.3	Model Validation Constraints	202
5.6	Failure Trace Data Set	203
5.6.1	Failure Trace Archive Data Structure	204
5.6.2	FTA Data Set Handling	206
5.6.3	Significant Data Set Parameters	209
5.7	Validating ISARC Configurations	214
5.8	Sampled Data Set for Fixed FTA Measurements	217
5.9	Sampled Data Set for Fixed Measurements using Applied ISARC Variations	219
5.9.1	ISARC Data Set Generator	220
5.9.2	Decision Model Parameter Handler	222
5.10	Results Summary	226
5.10.1	Result Set A Summary	227
5.10.2	Result Set B Summary	235
5.11	Chapter Summary	245
6	Conclusion	247
6.1	Thesis Contributions	251
6.2	Further Work	254
6.3	Closing Statements	257

A	Additional System Considerations	259
A.1	Additional Considerations	259
A.1.1	Dependability Characteristics	259
A.1.2	Reliability Constructs	261
A.1.3	Random System Behaviour	262
A.1.4	Failure Analysis in Distributed Systems	264
A.2	Node JS Signalling Constants	271
A.3	A Brief Overview of Systems Control using PID	273
A.3.1	Modelling Control Architectures with PID	274
A.3.2	Insights on Control Parameters	278
A.4	State of the Art Control Model Representation	279
B	Additional System Parameters for Runtime Configuration	282
B.1	ISARC Candidate System Parameters List	282
B.1.1	Installing Additional Architecture Parameter Choices	283
B.2	Additional Use Case Overview	288
B.2.1	Device Discovery - IoT Architecture Reference Model Overview	289
B.2.1.1	Type 1: Automated Device Discovery and Joining	289
B.2.1.2	Type 2: Automated Device Discovery and Partially Auto- mated Joining	291
B.2.1.3	Type 3: Manual Device Joining	292
	References	294

List of Figures

1.1	A Simple IoT Systems Architecture	4
1.2	A Preliminary Object Diagram for an IoT Protocol Set	7
1.3	An Extended IoT System Architecture	8
1.4	A Conceptual Overview of the Thesis	11
1.5	Thesis Core Concepts	13
1.6	Literature Summary	22
1.7	Machine to Machine Domain	27
1.8	Machine to Cloud Domain	27
1.9	Machine to Gateway Domain	28
1.10	ISARC System Scope Overview	28
1.11	Approach and Limitation Overview	32
2.1	An Aggregated Architecture Options Overview	45
2.2	A System Architecture Decomposition	47
2.3	Design Space Exploration Relationships	52
2.4	An Embedded System's Design Components	53
2.5	The SIMILAR Process	54
2.6	Model-based Quality Attributes	57
2.7	Model Options for Validation and Verification Activities	58
2.8	System Boundary Overview	59
2.9	A System Model for MBSE	61
2.10	A Typical State Machine	65
2.11	A Mobile IoT Cloud Reference Architecture	71
2.12	Android Layered Sensor Architecture	73
2.13	A Logical Network Deployment for Edge Connectivity	75
2.14	Web to IoT Protocol Transitions	76
2.15	Configuration Parameter Variability Characteristics	81
2.16	Configuration Architecture in an End-2-End Software Radio System	84
2.17	A Simplified Layered Architecture for System Configuration	85

2.18	A Systems Hierarchy Overview	86
2.19	System Components in Series	87
2.20	System Components in Parallel	87
2.21	Whole Product System Impact for Components in Series	89
2.22	System Form Overview - Configuration Requirements	90
2.23	Functional Details of the IBM MAPE Autonomic Manager	92
2.24	A Simple Process to DSM Summary	98
3.1	OPM Symbol Toolbar	105
3.2	OPM Function-Structure	107
3.3	System Requirements Extraction Layers	108
3.4	Definition of a Systems Architecture	114
3.5	IoT-Ce System Model Realisation Overview	115
3.6	A Mobile Cloud IoT System Domain for definition-to-OPM Extraction	126
3.7	Non-blocking Application IO	127
3.8	Samsung Galaxy running NodeJS in Termux	128
3.9	ED to GW Boundary Mapping	129
4.1	System Purpose:SD IoT System Value	139
4.2	Primary Process:SD1 IoT Edge System Value Delivering	141
4.3	In Zoomed Process:SD1.1 IoT Data Transmitting	144
4.4	SD1 Extension for System Value Environmental Control Configuration	149
4.5	In Zoomed Process:SD1.1 IoT Data Transmitting Environmental Control Candidate	150
4.6	Unfolded Structure:IoT System Controller Candidate	151
4.7	Analysis of Node JS Signalling Constants	154
4.8	OPD Unfolded Constant Notify Structure	156
4.9	OPD Unfolded Constant Request Structure	157
4.10	Gerund Feedback Loop Process State Transition	159
4.11	Function:ED System Request Handling SD	163
4.12	OPD:ED System Request Handling SDx	164
4.13	System Configuration Decision Model	169
5.1	System Model Overview	177
5.2	An Unfolded Front-end System Set	179
5.3	Function:OPD Device Addressing	181
5.4	Function:OPD Device Discovering Function	186

5.5	Sample Device Concept Summary	190
5.6	Sample Device Interrupt Form-Function Overview	190
5.7	NodeJS SIGINT Emitter Output	191
5.8	IoT System Controller Running Test Step 1 to 3	194
5.9	IoT System Controller Running Test Step 1 to 3 Alternate Initiation Point . .	195
5.10	System Model Extension Summary	196
5.11	FTA Data Structure	205
5.12	Standard Sample Population Simulation	213
5.13	A Conceptual Overview of the Thesis	214
5.14	System Configuration Decision Model to Code Summary	215
5.15	Decisions Based on Structural Integrity of System - Result 1	225
5.16	Results Summary Overview	227
5.17	Results Summary A Overview	234
5.18	Results Set B Sample Summary Overview	238
5.19	A ISARC modified PI Control Implementation	244
5.20	Results Summary B Overview	245
6.1	Integrated Thesis Contribution	253
A.1	State Escalation Flow	262
A.2	A Standardised Chart for Time to Failure	264
A.3	Failed State Occurences for DNS Server Sample	267
A.4	FTA DNS Event 7 State Change Summary	270
A.5	Conventional Control System Variables	275
A.6	Closed and Open Loop System Flows	276
A.7	Simulation Form-Function Definition	278
A.8	OPD for Control Foundations	281
B.1	Sampled Random Parameter Distribution for 52-Parameter Array Table . . .	288
B.2	A simplified layered architecture for IoT system interaction	289
B.3	Interaction Pattern for Automated Device Joining	290
B.4	Interaction Pattern for Semi-automated Device Joining	291
B.5	Interaction Pattern for Manual Device Joining	293

List of Tables

1.1	Master Reference List	24
2.1	Internet of Things Architecture Outlook	70
2.2	ED Operating System Growth Summary	72
2.3	Protocol Non-Functional Decision Summary	76
2.4	Capability-Protocol Decision Summary	77
2.5	Standards, Frameworks and Guidelines for Configurable Systems	91
2.6	A Simple DSM Process Equivalent	99
3.1	OPM Toolbar Usage Sample	106
3.2	A Limited Systemic Conception of IoT Connected Edge Awareness	109
3.3	Adapted Cyber Physical Gap Process Interaction	110
3.4	IoT Requirements Extraction Parameters	117
3.5	IoT-Ce Extracted Domain Parameters	119
3.6	IoT-Ce Front-end System Set Extracted Parameters	120
3.7	IoT-Ce Front-end System Set Process/State Extracted Requirement Parameters	121
3.8	IoT-Ce State Transition Extracted Requirement Parameters	122
3.9	IoT-Ce Control Extracted Requirement Parameters	123
3.10	Consolidated Form-Function Interaction	125
3.11	A Mobile Cloud IoT Extracted Requirement Parameters	126
4.1	IoT-Ce Form Function Adaptation	138
4.2	ISARC Parameter Sample	143
4.3	ISARC Supporting Parameter Sample	146
4.4	Node JS Signalling Notify Constants	155
4.5	Node JS Signalling Request Constants	157
4.6	Process Activity State Transition DSM	161
4.7	Process Activity State Transition DSM Clustered	161
4.8	Proposed Request Interactions for ED	162
4.9	Decision Model 1 for IoT-Ce Domain Interactions	170

4.10	Decision Model 2 for IoT-Ce Interactions for the ISARC SD through SD1.1	171
4.11	Decision Model 3 for IoT-Ce ED/GW Operating System Signalling	172
4.12	Decision Model 4 for IoT-Ce Process Interactions	173
5.1	DSM:Device Addressing Decision	182
5.2	CPS Logical-Function Gap Characteristics	185
5.3	ED Enrollment Error Set	185
5.4	DSM: Device Discovering Decision	187
5.5	Sample Device Parameters	189
5.6	IoT System Controller Test Step 1 to 3	193
5.7	IoT System Controller Test Step 1 to 3 Alternate Initiation Point	194
5.8	DNS Availability Event Trace Structure	206
5.9	DNS FTA Data Set First 7 Records	207
5.10	FTA DNS Significant Range	211
5.11	Codified ISARC Implementation and FTA Import Summary	216
5.12	Orginal Simulation Results for Set A: Population 10000	230
5.13	ISARC Result Set Field Descriptions	231
5.14	Simulation Results for Set A: Population 10000 & 20000	232
5.15	Simulation Results for Set A: Population 10000, 20000 & 50000	232
5.16	Simulation Results for Set A: Population 50000 FTA 384991	233
5.17	Simulation Results for Set A: Target Overview	234
5.18	ISARC Decision Model Impact for PI Variation	244
A.1	Random Sample Logic	263
A.2	Probability Mass Function Summary	263
A.3	Failure Model Analysis Methods	265
A.4	Summary of FTA Data Sets	266
A.5	Original FTA Data Set Event 6 to 8	269
A.6	Transformed FTA Data Set Event 6 to 8	269
A.7	Consolidated Node JS Signalling Constants	272
A.8	IoT Device Operating System Signalling Template	273
A.9	Control Model Validation Parameters	278
B.1	Consolidated Full Form-Function Interaction	284

List of Code Blocks

4.1	IoT System Purpose OPL	140
4.2	IoT System Primary Process OPL	141
4.3	ISARC System Realisation Flow. Adapted from Figure 3.5	142
4.4	IoT Data Transmitting SD1.1	144
4.5	Supporting Process:SD1 IoT-CE Value Delivering Updated Configuration . .	149
4.6	Unfolded Structure:IoT System Controller Candidate	152
4.7	OPL Unfolded Constant Notify Structure	156
4.8	OPL Unfolded Constant Request Structure	157
4.9	OPL:ED System Request Handling SD	163
4.10	OPL:ED System Request Handling SDx	164
5.1	OPL:Device Identifying Function	181
5.2	Function:OPL Device Discovering Function	186
5.3	Function:OPL System Process Interrupting	190
5.4	NodeJS OS Constant Emitter	191
5.5	Code Block 1: ISARC Decision Model 1 Initialisation	198
5.6	Code Block 2: ISARC Decision Model 2 Initialisation	199
5.7	Code Block 3: ISARC Decision Model 3 Initialisation	200
5.8	Code Block 4: ISARC Decision Model 4 Initialisation	201
5.9	Failure Trace Event Pandas Import	207
5.10	Code Block 5.1: DNS Event Trace Transformed Import	207
5.11	Code Block 5.2-DNS Event Trace Clusters	208
5.12	Code Block 5.3-DNS Event Trace Most Active Nodes	209
5.13	Code Block 5.4-DNS Event Trace Pivots	210
5.14	Modified Sample Population Distribution	212
5.15	_GlobalVariableHandler	216
5.16	_GlobalEventSampling Summary	218
5.17	DNSEventClusterSummary Validation	219
5.18	_GlobalDataSetGenerator	220

5.19	ISARC Decision Model Primary Data Set	222
5.20	ISARC Decision Model Manual Setting of Selection Modifier	224
5.21	ISARC Decision Model Simulation Output	224
5.22	Front-end System Set Communication Interaction Sensitivity Value	226
5.23	Front-end System Set Communication Interaction Sensitivity Value Modified	226
5.24	Results Set A Sample Output	228
5.25	Original Simulation Results for Set A: Target Table Realisation	235
5.26	Result Set B: Fixed ISARC Decision Model Output	236
5.27	ISARC Decision Model Control Functions	240
A.1	Expanding FTA Time Array Records	269
A.2	Control System Foundation in OPL. Extracted from Dori (2002)(pp.393) . . .	280
B.1	ISARC OPM Parameter List Assignment	283
B.2	Additional ISARC Parameter List Assignment with Random Generated De- cision Sensitivity	285
B.3	ISARC Sampled Parameter Selection Averages	288
B.4	Manual Device Joining in OPL Form	292

Chapter 1

The Evolution of Things

1.1 Introduction

The Internet of Things (IoT) claims to connect an array of devices into an interconnected system of systems. Across the spectrum of IoT research, functional systems coverage supports an appetite for heterogeneous technology advancement. The growth of interest in IoT has become such a topical study that we can barely see the wood for the trees. Finding system value in this vast body of knowledge is no different. In modern research, the IoT body of work mainly applies domain and functional coverage related to the visualisation of its architecture. This visualisation aims to argue for connectivity, protocols, and elusive killer IoT applications. What is missing is a sound approach to tackle the view of IoT for what it is - a system with form, function, and value.

In tackling the problem using a model-based approach, the thesis proposes a systems architecture to outline the systems, subsystems, and components employed at runtime in the IoT. The Object Process Methodology (OPM) ISO-19450 guides the modelling of the systems architecture through strict notation usage and principled modelling activities. Care is taken in applying the OPM ISO-19450, as with most architecture modelling tools, the usage is generally part creative, part structured. The **IoT-Ce Systems Architecture for Runtime Configuration (ISARC)** suggests a hierarchical structure for a multi-layered connected heterogeneous IoT system.

The thesis also provides a method to validate the ISARC using OPM and Design Structure Matrices (DSM). The output of the model-based approach aims to position a repeatable method for extracting knowledge from broad and unstructured IoT claims. The decision model applies both qualitative and quantitative theories against the data set achieved by extracting architectural decision parameters from selected IoT references.

1.2 Background

The evolution of embedded systems has reached a point where these systems touch almost every part of our daily existence. With an ever-increasing need for technological advancement, these systems will become the physical extensions of the internet. This cyber-physical evolution goes beyond the services that individual embedded devices expose. It underpins the IoT by creating connected edge systems that integrate large-scale digital and physical processes. Although the precise definition of IoT is particularly subjective, some key components support the definition as an interconnection of “unique” sensors and actuators into a broad communications network, as seen in IEEE (2015). Due to disparate spatial and temporal requirements, the IoT requires fundamentally different systems management capabilities. These capabilities can play a crucial role in the differentiation of measurement within a system under dynamic conditions.

The IoT body of work covers a broad research agenda that outlines increased interaction levels between the structures and functions of components, subsystems, and systems in the IoT ecosystem. Increases in interaction due to evolving end-user requirements are a natural progression for any system. In IoT, many standards, interaction requirements, and form constraints create massive interaction possibilities due to their different deployment options. IoT research agendas introduce a functional classification of overlapping concerns. As seen in Vermesan et al. (2009), Vermesan et al. (2011), Ailisto et al. (2011), Miorandi et al. (2012) and Stankovic (2014), these research agendas argue a baseline for an ever-changing IoT ecosystem containing single or multi-capability compositions. The outcome that the extensive body of IoT work produces primarily suggests that *Things* will interact and requires some form of communication. In some view, the IoT sees seemingly unrelated objects establishing communication and ubiquitously sharing data on a large scale as shown in Bassi et al. (2013), and European Commission CORDIS (2013). As recent as 2011, IoT has been viewed as an extension of embedded systems, thus inheriting embedded systems characteristics, Kopetz (2011). The characteristics that suggest the likeness of IoT to embedded systems confirm the view on device uniqueness, interaction, and design practice that informs effective systems management activities. These are;

1. **for simplification** in resolving human cognitive constraints as seen in Kopetz (2011), and enabling of mechanisms to cope with complexity using configuration, Ailisto et al. (2011). Additionally, the use of co-design between software and hardware to reduce inconsistencies is also a known juncture between IoT and embedded systems, Stankovic (2014).

2. **for greater usability** by focusing on a population that contains usable components such as mobile devices, Kopetz (2011), and those used in discovery and closed-loop communication, Stankovic (2014).
3. **for reuse** through the use of existing standards to meet the needs of IoT use cases, Kopetz (2011). For example, the use of conventional resolvable addressing protocols in Domain Name Systems supports Object Naming Services for edge-device identification, EPCGlobal Inc. (2008). Additionally, the reuse of internet protocol-based interaction standards such as web services communication using the Constrained Application Protocol (CoAP) is also a known IoT standard, Bormann et al. (2014).
4. **for management and optimisation** as indicated by Kopetz (2011), by guiding smart object interaction across dimensions such as state and policy awareness, Vermesan et al. (2009).

The body of research for IoT presents an ecosystem in its entirety. The IoT application subset can broadly be seen as a connected system, containing three dominant domains of focus; *data generation & acquisition*, *data transmission*, and *data analytics*, Chuah (2014). Notwithstanding other system themes such as scalability, integration, and security, research into each of the dominant IoT domains attempts to meet many research goals. These goals range from architectures for end-to-end energy efficiencies of devices that deliver core IoT sensing capabilities (*generation & acquisition*) to critical decision-making possibilities found in the data present in the sensing process (*data analytics*). This *generation & acquisition* and *data analytics* range is evident in the views by Kaur and Sood (2017) and Shaikh et al. (2017), and Qiu et al. (2013), respectively. The delivery of a solution to a key IoT problem statement is an exciting challenge that faces IoT researchers, as the body of work for IoT stretches across non-critical and critical service delivery. Due to vast structural and behavioural variation, these domains produce high complexity levels as expected in systems with many integrated components and interactions. This view on complexity is a generalisation as applied to systems architecture conceptualisation by Dori (2011), and in complex systems architecture study in Crawley et al. (2016).

The three dominant domains of focus position IoT architecture to become the next archetype for lightweight systems integration. For this reason, it overlaps into areas of cloud computing and big data, which have made their way into IoT research. The overarching requirements for the management of systems that process large amounts of data reflect similar architectural components as required in an IoT application ecosystem, as seen in Taleb et al. (2015). This requirement is primarily due to the volume and variety of data flows between

applications in an IoT application ecosystem. Thus, increasing interdependence between heterogeneous IoT application system components.

IoT applications play an essential role in the architecture at the center of these domains as

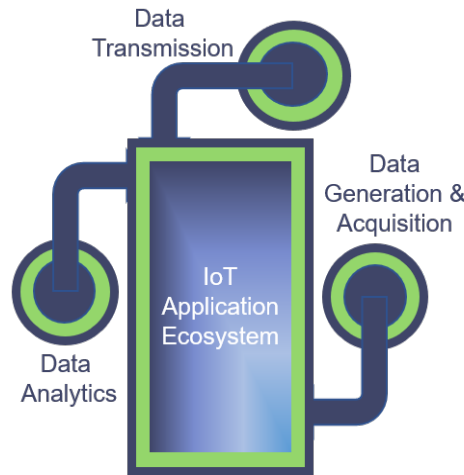


Figure 1.1: A Simple IoT Systems Architecture Perspective adapted from Chuah (2014)

these applications have had the most exposure in the visible IoT project domain. A further specialised definition of this simple architecture indicates a cross-cutting definition of physical and digital convergence. IoT applications implement the structural and behavioural aspects of the IoT, as indicated in Vermesan et al. (2011). Figure 1.1 outlines a systems architecture perspective that concurs with the representation of a convergence of at least three domains indicated in Vermesan et al. (2011). These are;

1. technologies such as nanoelectronics, sensors, actuators, embedded systems, cloud computing, software,
2. communication and
3. intelligence.

Globally, consumer applications have shown success, but there have also been occurrences of large-scale failures despite these successes. In the local South African market, devices with secondary IoT functions have unfortunately failed to launch due to broader product strategy failures, as seen in Mcleod (2014) and van Zyl (2015). This failure exhibits the relative immaturity of IoT strategies, which reflect the broad scope of industry functional requirements, such as those seen in Gartner (2017). In the vertical application areas where standardization is missing, these failures are due to fragmentation and poor compliance, IEEE (2015). However, there have been successes in IoT for companies in South Africa, such as IBM and MTN. These use cases produce a more direct system purpose, such as

wildlife protection, IBM (2017).

Companies such as IoT ONE have produced a vast number of use case variations across multiple industry verticals. Verticals such as transportation, automotive, equipment, and machinery industries have seen a fair amount of case studies, IoTOne (2019). Movement closer to end-user controlled devices in these ecosystems is known as the edge, as indicated by Yu et al. (2017). This architecture requires a structure that balances the characteristics of each of the ecosystem's subsystems given known constraints such as power, processing, and communication capabilities. In broader industrial terms, the sensitivity of power, processing, and communication decisions can be the difference between life and death. In line with a natural progression of consumer technology convergence due to increasing consumer needs, a move towards systems generality underpins its growth on the availability of readily available sensor technologies found in generally available mobile devices. The most prevalent is the vast possibilities of combining capabilities in mobile devices to serve a greater utility. Mobile devices in the IoT body of research manifest as a vital edge device that services human needs - a constrained device and gateway - to be more precise. Furthermore, Pei et al. (2013), Horn et al. (2016), Zhang et al. (2017), Sinche et al. (2017), and Khazbak et al. (2017) present use cases for mobile device-based IoT. These use cases focus on individual human-centric system purpose, as opposed to an industry-centric purpose. The increased usability of mobile devices moved to the edge supports the proposed move of IoT to establish a usable, connected edge. Finally, coupled with the mammoth increase of mobile device populations, as seen in IDC (2017) and IDC (2018), the requirement for treating edge devices as a vital resource should be highly significant.

The background on the evolution of things indicates that research agendas have become broad; however, the simplification of IoT-bound and systems-bound problem statements should be strongly considered based on the potential utility of IoT use cases in the future.

1.3 Overview of a Fragmented IoT Ecosystem

Forecasts for the number of connection-ready IoT devices lie broadly in the billions with increases over the next decade by billions more, Anderson and Rainie (2014). Additionally, claims on trillions of devices are also noted, Stankovic (2014). Although forecasts vary by IoT research body and IoT vendor, one common theme is that these devices will generate and transmit data on scales not seen earlier by man, or for that matter, by systems known today. Thus, management and control play an essential role in the architecture of these new

systems. The core systems viewpoint between what controller-based automation is historically and what IoT will be in the next decade is a stark realisation of the preparedness to handle the influx of rapid event-level data flows. Ancillary control of this ecosystem exacerbates inefficiencies for mass-market sensors and actuators served by general-purpose mobile devices. Thus, the question is not a case of why interest is necessary in IoT systems architecture, but rather how to efficiently make decisions about IoT parameters that enable primary functions and value through devices at the edge, given increases in fragmentation.

Thus, the basis of the problem identified for the thesis is a series of positions suggesting how to reduce complexity in a system composed of non-standard packaged sensors, actuators, and internet-connecting components. Having to redeploy or manage devices in a fast-moving pre-packaged application environment can become problematic due to applications' rapidly changing nature. IoT applications are underpinned by several standards, frameworks, with interoperability being somewhat loosely defined. As seen in IoT protocol sets, Postscapes (2017) outlines a grouping of similar protocol capabilities at each layer of an IoT system set. Although each of the protocols recommends a best-fit use case, the protocol variation and choice can become an open-ended architecture decision. For example, JSON-LD and the Web Thing Model cited in Postscapes (2017) offer different protocol specifications to specify an IoT system such that it becomes additive or incremental. Architecture decisions that drive protocol selection in the system could be due to protocol support, device support, and integration complexity. In Trifa et al. (2015), the Web Thing Model provides function principles, whereas JSON-LD (Linked Data) by Kellogg (2019) provides structural data guidances. With several loosely coupled standards in the IoT domain, a systems architecture provides a consistent way to model heterogeneous smart ecosystems conceptually while representing dependencies between these protocols. As a simple object model containing an aggregation relationship, the IoT Protocol is shown in Figure 1.2. This representation is a simplified view of a notation supported in modelling languages that supports aggregation. In line with a simplification and subsequently single notation requirements, the object model is shown using a structural representation from OPM ISO-19450. Recent work in model-based approaches for protocol specification is discussed by Mordecai and Dori (2014). However, the notation style agrees with most aggregation relationships similar to those found in UML and SysML.

The use of aggregation in modelling notations - and largely model representation - creates a reduced representation of reality. This view abstracts system problems and fragmentation into a state that closely aligns with the perception of what the model aims to represent. Within the understanding of a fragmented IoT systems reality, fragmentation risk increases

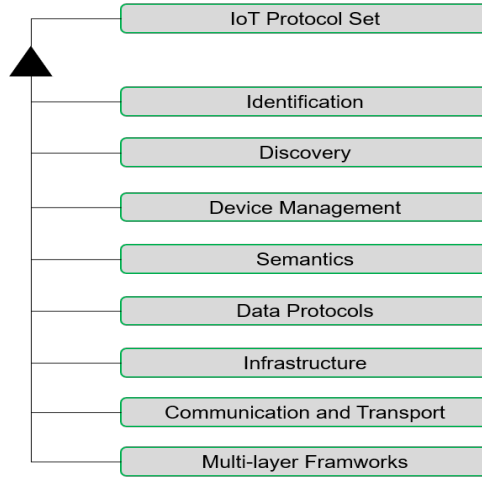


Figure 1.2: A Preliminary Object Diagram for an IoT Protocol Set representing Protocols in Postscapes (2017)

as the decomposition of the IoT ecosystem increases. However, abstraction can handle fragmentation concerns. This view is evident in general systems design using hierarchical structures in systems, Skyttner (1996)(pp.40). This risk materialises when decomposing Figure 1.1 into an extended capability set. The system purpose present in Figure 1.1 places devices in the edge capability set found in the EPC Global Architecture. The EPC Global Architecture, shown in Uckelmann et al. (2011)(pp.21), offers a futuristic capability overview for the IoT. In this architecture, a split capability model represents both core and edge architectures. Coupled with the guidance on reusing device capability, the device hardware such as onboard mobile phone sensors offers one of the most convenient sensor populations in circulation today that already serves the interfacing, acquisition, and control functions in the EPC Global Architecture. A prior assumption guided by recent IDC reports indicates an increase in the number of devices that have been distributed worldwide over the first quarter in 2017, reaching approximately 347 million units, as indicated in IDC (2017). The associated impacts of this distribution are that both lower and higher-cost devices are enabling more data acquisition and control possibilities into the end-user market. With a simple overlap between higher mobile device distribution and sensor capabilities, a more comprehensive architecture can be argued for. For example, the proposed extended EPC Global architecture supports the future of IoT by stipulating the classification of IoT capabilities, Uckelmann et al. (2011). Although Figure 1.3 contracts the core number of domains to only two domains - core and edge architectures - this representation positions a clear boundary between component, interface, control commands, and data. A view that extends the fragmentation problem in both core and edge boundaries of the architecture is the integration requirements for non-Internet Protocol (IP) devices. The architecture indicates

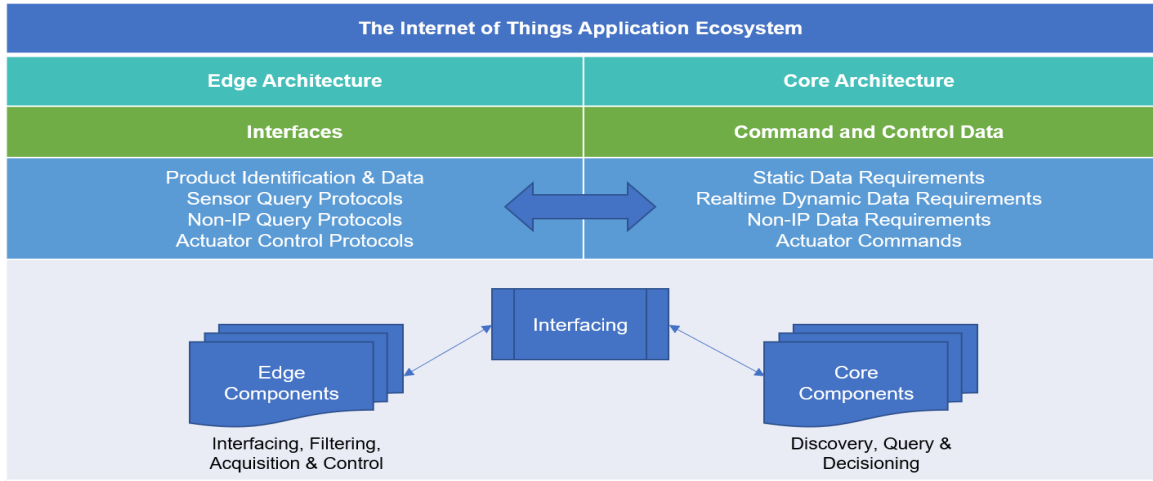


Figure 1.3: An Extended IoT System Architecture adapted from Uckelmann et al. (2011)

these to be low capability devices exposed via the IoT. Furthermore, the commands and data located on the boundary of the core and edge architecture in Uckelmann et al. (2011) present an opportunity to bridge protocols while still maintaining a level of independence between low-cost devices and more costly core application and infrastructure architectures.

Given the immense growth of device variation and connectivity possibilities, essential system boundaries, functional and structural constructs still require a logical representation to indicate the specific system's primary purpose. As seen in the split between edge and core architectures in Uckelmann et al. (2011), the core holds the main command-and-control capabilities, so an approach is still needed to manage a class of state notifications between components in the IoT edge system boundary. This split ensures flexibility while maintaining dependable connectivity within the constrained systems environment closer to the IoT edge.

1.4 Dealing with the Edge

With the reinvention of the IoT edge mobile device domain as a formal industry, these devices have spawned multiple commercial and hobbyist use cases. The influx of mobile use such as Sinche et al. (2017), Pei et al. (2013), Horn et al. (2016), Zhang et al. (2017) and Khazbak et al. (2017) can be seen in research. Where Sinche et al. (2017) and Pei et al. (2013) indicate the use of mobile phones for behavioural tracking, location and triangulation, the work by Horn et al. (2016), Zhang et al. (2017) and Khazbak et al. (2017) relate closer to mobile health care interactions. The enablement of these IoT use cases supports the need for core onboard sensor technologies packaged into a single mobile form.

An increase of mobile applications such as those seen on the Google Play and Apple Store drives a commercial agenda. The reinvention of hobbyist use cases is evident in the activity around Arduino, Raspberry PI, and Beaglebone microcontroller-driven user application prototypes on Instructables.com (2017), and crowdfunded initiatives, Chuah (2014). In each of these areas, embedded systems play an important role in delivering system value. In dealing with the edge, it is safe to indicate that one would engage the notion of an embedded system.

An embedded system comprises of components and devices that operate in a singular or multifaceted environment - designed to support a specific service or need. Embedded systems identify as “information processing systems” enclosed into larger products or systems, Marwedel (2014). These systems can link directly to the physical environment within which they operate and are thus further defined as “cyber-physical”, Marwedel (2014). The definition of embedded systems indicates integration between sensing and the sensed environments. It has been shown that the behaviour of a system under this integration might encounter an operating gap due to the nature of either end of the system - namely digital or physical ends. This is a systems architecture viewpoint on cyber-physical modelling, Mordecai and Dori (2017a). Examples of embedded systems are found in consumer products such as mobile phones, vehicles, or home automation solutions. The value of considering these embedded systems into the IoT value chain supports the notion of system boundary creation. This boundary is necessary when these consumer products are enhanced to serve other purposes, such as sensor-rich mobile devices for edge use cases, autonomous vehicles, and adaptive automation solutions.

The basis of the IoT generation & acquisition subsystem mentioned in Figure 1.1 is, in fact, a cyber-physical system (CPS), which contains devices that gather and communicate physical phenomenon. This claim’s reasoning is the inherent nature of digital systems, but more aptly, the duality of CPS as seen in Mordecai and Dori (2017a). In their view on cyber-physical existence, Mordecai and Dori present a natural gap between the cyber and physical worlds. A conceptual model presents a highly complex system as the basis of the modelling effort to prove the handling of the cyber-physical gap. Mordecai and Dori produce a position that not all arrangements of variables between the physical and cyber worlds can be mapped, thus producing gaps in observation and control actions. The systems presented in Mordecai and Dori (2017a) exist in physical and informational form. Thus, the interaction through the system’s physical form requires similar handling of the informational structure. Given the deep understanding of CPS in current research, recent advancements in creating a broader market for aggregating sensor capabilities via smart endpoint integration have

been recommended through interfacing theories by Lohstroh and Lee (2015) and Latronico et al. (2015). In dealing with the edge, a drive towards an interface theory represents an architecture that abstracts physical components into standards of web-based accessible components. This adaptation immediately lends itself to hardware-software co-design activities based on considerations for both hardware and software configurations in the delivery of system value. With devices becoming more pervasive, usable, and, interconnected, the need for a more efficient approach to modelling complex IoT systems is necessary - which operates within the constraints of the *Kopetz* principle.

In a view on systems design versus systems realisation, the Kopetz principle indicates that the properties of the system that a model asserts are properties of the model itself, Ptolemaeus (2014)(pp.10). In support of clarifying the representative properties of system complexity with suggested increases in model fidelity, the Object-Process Methodology (OPM) ISO-19450 presents a means to simplify unnecessary system architecture modelling complexities, ISO/TC 184/SC 5 (E) (2014). Thus, decreasing unnecessary assertions about model realisation in a real-world scenario. The OPM is an ISO conceptual modelling standard that views objects, processes, and states as primary constructs for systems modelling. The standard supports the modelling of well-structured systems architectures by modelling the structure and behaviour of a system using a single diagram and language notation. Simultaneously, the OPM drives completeness and clarity characteristics in model representation, Dori (2011). The OPM creates multi-purpose outcomes validated against model requirements using Object Process Diagrams (OPD) and Object Process Language (OPL) in a single systems architecture specification activity.

1.5 Thesis Objectives

The research questions are founded in unique design domains; **Systems Architecture**, **IoT**, and **Configuration**. These domains have variable levels of maturity in their literature, and hence an exploratory position will outline the extent to which convergence is possible between these domains. The conclusion of the argument on whether a systems architecture viewpoint on IoT-Ce runtime configuration impacts communication dependability is supported by individual vertical premises. These are; the existence of systems architecture as a domain, the application of well-formed systems principles into IoT, and the IoT-Ce. Finally, a parameter-based simulation aims to suggest a proof of greater determinism in architectural decisions for the IoT-Ce. Understanding that each of these domains has a unique history allows the study to position the limitations of the primary hypothesis. The hypothesis presents as;

“Low level IoT-Ce configurations that influence decisions about communication dependability can be abstracted into a conceptual IoT system architecture in order to support model validation activities”

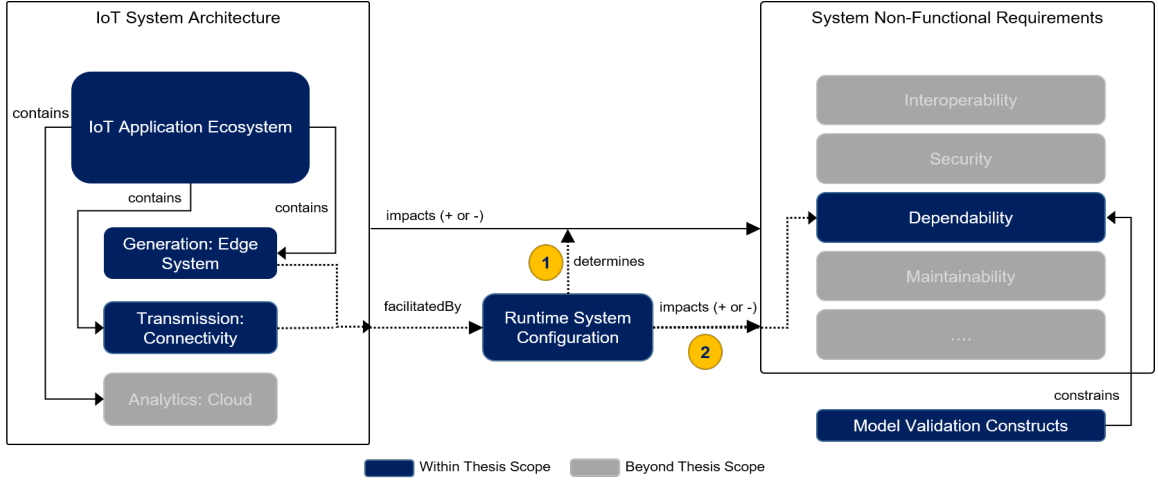


Figure 1.4: A Conceptual Overview of the Thesis

Figure 1.4 represents the objectives of the study. These are;

1. to propose a standards-based systems architecture that outlines the runtime configuration of IoT application components found in the IoT-Ce, and
2. to enable the generation of a data set using the the model’s output, and allowing model tuning using a novel control strategy in line with communication dependability impacts.

As a result of increasing system heterogeneity and the need for greater control flexibility at the IoT-Ce, the research questions are thus defined as;

1. How does the runtime configuration of IoT system, subsystems and components support dependability though reliable state awareness of the IoT-Ce?
2. How can different architectural options contained in a heterogeneous system be modelled to drive minimised error and control actions for the IoT-Ce?

This thesis presents two contributions. Firstly, the study proposes a standards-based systems architecture method that outlines the systems, subsystems, and components employed at runtime in IoT edge systems. The Object Process Methodology (OPM) ISO-19450 systems architecture guides the modelling of the systems architecture. The **IoT-Ce Systems Architecture for Runtime Configuration (ISARC)** suggests a hierarchical structure

for a multi-layered connected heterogeneous IoT edge system. The study of the runtime properties of systems can provide a better understanding of how to model systems that are dependable and scalable. Additionally, these systems have the requisite properties to find application in large-scale distributed systems design. The primary contribution is the suggestion that identifying well-defined system, subsystem, and component boundaries in the IoT-Ce applies to a known approach supporting complexity reduction. The proposed approach provides the method for testing variations of configuration for the IoT-Ce. The ISARC succinctly places the definition of configuration outside of the realm of self-adaptation or reconfiguration, and additionally, is not as far entrenched into pure configuration management. The ISARC argues for pre-existence as a primary construct in IoT-Ce systems architecture.

Secondly, the thesis provides a method to validate the ISARC using OPM template design and Design Structure Matrices (DSM). The method for modelling configuration decisions for the IoT-Ce uses OPM as the basis for parameter extraction and DSM to represent the architecture. The method applies an existing body of knowledge using these standards to deliver well-bounded objects, processes, and states. A set of suitably bound decision parameters supports a test of conceptual architecture variations of the ISARC using a novel concrete control design based on instantaneous and accumulated errors. This contribution's primary benefit extends the systems architecture by aiming to prove that the ISARC can support ongoing runtime configuration requirements using a series of simulated data set samples using Python. A series of data generation code blocks output the result of interaction decisions between ISARC model components. Finally, these samples provide a comparison to a historical data set that mimics a hierarchical systems architecture found in a failure trace data set for a multi-tiered domain name system (DNS).

In summary, the primary objectives of the thesis are to;

1. propose a systems architecture that applies symbolic conformance to OPM ISO-19450,
2. model a single systems boundary (IoT-Ce) within a larger complex system (IoT),
3. argue the requirement to understand pre-existing runtime arrangement of the system, in order to support adaptive properties.

A consideration that supports the primary objective is to;

1. propose a compact decision model that houses interaction decision tradeoffs by proposing a method for ongoing configuration decisions for the IoT-Ce.

1.6 Literature Overview

The proposed research study’s goals are motivated by the need for systematic studies that outline the impact of variations of conceptual system architecture configurations for abstracted mobile devices that form the device population in the IoT-Ce. A primary output of the study creates a systems architecture stipulating a structure and behaviour in the IoT-Ce. In keeping to a structured systems design, the proposed literature also conforms to a vertical domain viewpoint. Thus, the core themes approach individual primary areas of concern and then integrate these into a consolidated primary-secondary function that supports the core hypothesis. Figure 1.5 represents the core themes that form part of the thesis. Understanding the primary themes in these vertical domains is necessary because it

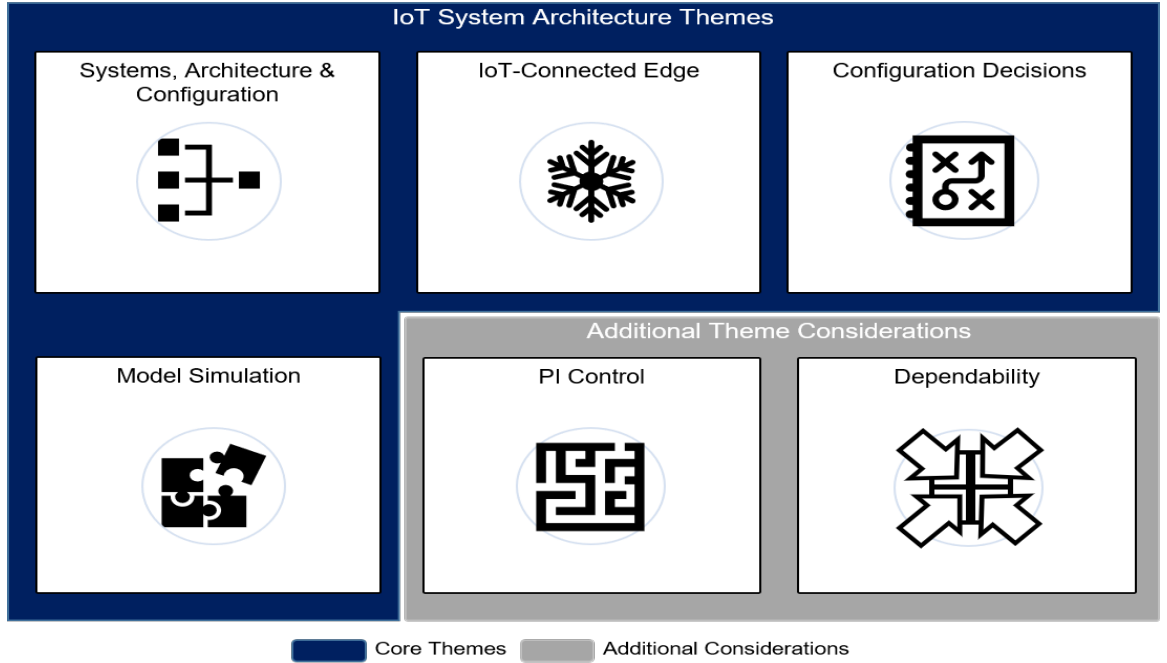


Figure 1.5: Core Concept Overview for the ISARC

drives the core argument through the analysis of each primary value that individual literature assessments exhibit, before integration into consolidated arguments and evidence. This approach is necessary based on the varying maturity of each core theme. The literature overview sections that follow, outline the core considerations for each theme.

1.6.1 Systems Architecture

In a view on distinguishing between architecting, engineering, and project management, Maier and Rechtin (2009)(Preface xvii) indicate systems architecture as identifying outside the realm of systems engineering. Where systems engineering applies deductive reasoning,

systems architecture exhibits non-deterministic principles during design. Maier and Rechtin (2009) states this as being generally unmeasurable using non-quantifiable tools and based on experience. Systems architecture is largely inductive by nature. Engineering is seen as a demanding outcomes-based field, whereas architecture manifests as more of an art, Maier and Rechtin (2009)(Preface xvii). Despite these constraints, the creation of a well-formed systems architecture still adheres to principled design. Principles and laws can still govern the output of structured systems architecture. For example, principles that guide system structure through sum-of-part thinking are system holism, suboptimisation, and darkness principles, Skyttner (1996)(pp.61). Holism applies the principle that system value is attained by the whole and not necessarily any individual parts. Additionally, guarding against suboptimisation ensures that each subsystem value is regarded as part of the whole and not separately. Finally, the darkness principle applies that no system can be known entirely.

Furthermore, more robust laws of operation such as the Law of Requisite Variety and Weinberg’s Complementary Law drive core system thinking methods in applying control and model perspectives. Finally, The Law of Requisite Hierarchy indicated in Skyttner (1996)(pp.61), states that “the weaker and more uncertain the regulatory capability, the more hierarchy is needed in the organisation of regulation and control to get the same result.”

The principle of hierarchy underpins the specification of systems in general systems theory, not only seen in Skyttner (1996) but also in Dori (2002). Thus, the placement of system components within this hierarchy aids in clarifying the structure and dynamics of the system under specification. Furthermore, the modularisation of components into subsystems, and subsystems into systems, enables a set of boundary questions for the systems architecture. The belief in systems is that a multi-levelled hierarchy exists. This hierarchy represents static structures to simple dynamics, to self-regulating behaviours, and then finally to a system set that determines its end state via the understanding of “absolutes” that exhibit systemic behaviours. This hierarchical view is noted in Skyttner (1996)(pp.71) and Dori (2002)(pp.386). Due to the incremental structural form in systems, a set of pre-existence criteria find their way into systems architecture. Pre-existence characteristics within systems architecture should;

- reduce complexity, increase model fidelity and trust, and drive the importance of state awareness by representing a natural order of systems purpose. Furthermore, *dual-channel processing*, *channel limitations* - which entails using objects that convey specific facts about the model - and *active learning* during system specification drive

the primary simplification and general usability of systems architecture models. These characteristics present Mayer's Cognitive Assumptions cited in Dori (2011).

- support a path towards system specification for dependable systems in a changing heterogeneous ecosystem necessary for specific classes of IoT systems. In the real-world implementation of these systems, the characteristics of what a model represents and the actual implementation play a vital role. As stated by the Kopetz principle in Ptolemaeus (2014), the attributes that the model exhibits represent the model itself and not necessarily the choice of realisation of the model.
- decompose and modularise applications that can play an important role in variations of IoT use cases. Understanding usable and modular applications could be one of the key drivers of the IoT's success. This architectural approach can effectively spread implementation risk through lower implementation costs. Reference design variations such as those seen in Arrow.com (2018), can further offer a means to create usable modular IoT designs in IoT systems architecture. Modular systems approaches can be seen in the processes that reduce complexity using systems engineering for complex systems. A high cardinality of interconnections, functional aspects, and form constructs in the system specification defines its complexity. This view of complexity is held by Dori (2011) and in complex systems architectures, Crawley et al. (2016)(pp.49).

Research shows that historical systems modelling approaches can be used to produce systems architectures; however, the time to show value with these methods increases due to the broad reach of stakeholders that will require different diagrams, Grobshtein and Dori (2008) and Al-Fedaghi and Alahmad (2017). Ordinarily, these models produce different value to its reader and model user. However, a consistent lineage of requirements needs additional modelling constructs, such as matrix-based requirements traceability models. Research also shows that specific use models such as state machines is not well understood, Wagner and Wolstenholme (2004a), Robinson (2004) and Wagner and Wolstenholme (2004b). Apart from being one of many model types in historical systems modelling approaches, the core use of state model outputs is, in fact, a secondary function in these modelling approaches. State of the art modelling approaches such as OPM proposes the importance of object, process, and state as primary model drivers, ISO/TC 184/SC 5 (E) (2014). The OPM is already used to model complex systems such as those seen in Mordecai and Dori (2017a), de Matos (2018) and Li et al. (2019).

1.6.2 IoT Connected-edge

The IoT body of knowledge applies a broad research agenda. This research agenda creates a set of functional components, subsystems, and a series of capability coverage statements. These research agendas cover the primary goals of networking, communication, energy management, configuration, interfaces, identification, discovery, processing, interoperability, service enablement, user adaptation, vertical industry integration, Ailisto et al. (2011).

IoT agendas are usually coverage rich to ensure that overall IoT solutions meet one or more system goals. Characteristics of the core smart device within an IoT ecosystem generally tier into what these smart devices are and what they are not, Vermesan et al. (2009)(pp.9). In Vermesan et al. (2009)(pp.9), the authors indicate fundamental aspects of things as being real or virtual with the requisite information exchange between them using protocol-bound communication. Furthermore, environmental interaction, the autonomous nature of interaction, and intelligent self-replicating functions allow the creation, destruction, and management of smart device representations. In Vermesan et al. (2011)(pp.13), the authors indicate these dimensions as a set of interacting behaviours that intersect representation, interactivity, and awareness dimensions. Where representation reflects the function, behavioural rules, and workflows of a device, the awareness aspect of devices focuses on the dynamic handling of device behaviour. These behaviours stretch across activity, policy, and process awareness. Process-aware behaviour creates higher interactivity for mature representations of devices. Therefore, a set of cross-purpose system goals drives interaction and connectedness in an IoT ecosystem. Further views on capability coverage can be seen in system characteristics such as heterogeneity, energy optimisation, data exchange and self-organisation as seen in Miorandi et al. (2012), Stankovic (2014) and IEEE (2015). Finally, references to standards-based approaches to the aforementioned capabilities is a clear research direction as seen in Miorandi et al. (2012), Stankovic (2014), IEEE (2015), and Gartner (2017).

The expansion of IoT capabilities has given rise to capability reduction representation using core systems theory approaches. In general systems theory, the layering of architectures and components delivers a cleaner representation of functional boundaries. Applying greater structural splits in the core functions of the IoT can be seen in several frameworks such as Uckelmann et al. (2011)(pp.21), Bassi et al. (2013)(pp.168), Sridhar (2013), and Chuah (2014). These frameworks also give rise to a split of IoT architectures into two key areas; The core and the edge. The edge delivers control close to the end-user of the smart device. Thus, the end-user ability to access IoT services without unnecessary friction in the smart device interaction processes drives system value and system purpose. Frameworks

that enable greater edge control is seen in research by Yu et al. (2017), Jain and Mohapatra (2018)(pp.19) and Chang et al. (2019)(pp.13). The primary goals for this control relate to energy efficiencies, optimised traffic profiles, seamless access, and shifting of intelligence to the edge. Given the architecture that underpins the movement of these goals into an edge system boundary, the availability of a large population that already exists in this boundary will serve the purpose of agent-instrument relationships and interaction. Primary systems formation questions already position agent-instrument interaction as a core driver for modelling systems. Systems modelling questions and guidance is seen in work by Dori (2011), and Crawley et al. (2016).

The availability of generalised sensors and control is a known architecture in mobile instruments today. Research indicates that an increased number of mobile edge devices can help realise the IoT-Ce value by increasing sensor populations. Thus, making statements that relate to billions of IoT devices plausible. The variation of function and interconnectivity of these devices provides a suitable basis against which the IoT's complexity is true - for some time. This complexity exists for both software, hardware, and combined software-hardware systems. The evidence to support this is founded on large codebases for mobile device control and applications. These IoT applications support basic human needs where device communication is needed and abstraction activities where physical components are virtualised. Increases in device usability using massive device populations support increases in end user system value as seen in Pei et al. (2013), Horn et al. (2016), Sinche et al. (2017), Zhang et al. (2017), Khazbak et al. (2017) and EvaTech (2019). Furthermore, the growth in device populations with varying onboard sensing capabilities is an indication that the edge will continue to expand to offer true systems value to edge users. Device growth and use case variation is a known factor in mobile architectures that support IoT in current times, as seen in Instructables.com (2017), IDC (2017), IDC (2018), IoTOne (2019), Arrow.com (2018) and GSMArena (2018).

For smart mobile devices that represent the core instrumentation in the IoT-Ce, the implementation of systematic test-ready modelling approaches and patterns consider these instruments as hierarchical and heterogeneous. This implementation approach is due to the variation and number of IoT-Ce use cases present in academia and industry. One such approach is the use of actor-driven accessor patterns. An actor is a software component that implements a digital interface against which IoT protocols integrate, Latronico et al. (2015). The actor pattern implements some of the core protocol groups in Figure 1.2. The benefit of using an accessor is that it wraps actors into electronically accessible units, which expose

heterogeneous connected edge device (ED) capabilities over a series of standards-based protocols. A similar viewpoint holds for non-IP versus IP-based integration in frameworks such as the EPC, Uckelmann et al. (2011). The practical nature of the accessor pattern makes it a suitable component in the end-state architecture as it realises an implementable set of capabilities for data access and state-awareness in the IoT-Ce. Furthermore, the abstraction of EDs could meet requirements for wrapping non-IP-based devices, as indicated in Figure 1.3. Thus, abstraction of physical devices and access to physical device resources through the exposure of device capabilities using digital methods is seen as state of art in Marwedel (2010), Lee et al. (2012), Seshia and Lee (2012), Marwedel (2014), Brooks and Lee (2017), Latronico et al. (2015), and Lohstroh and Lee (2015).

1.6.3 Runtime Configuration

The definition of configuration in software engineering indicates the arrangement of components and interconnections between those components, IEEE (1990). From a primary systems point of view, as indicated in the laws that govern system theory in Skyttner (1996)(pp.61), the arrangement of components into a whole is shown to indicate a composition of lower level elements into higher-level subsystems and then finally into a broader system. The concept of system, and its intended purpose, is dependent on an observer that gains value from applying the use of the system. A priori condition for system value is noted as the existence of a human-device, or agent-instrument, interaction as seen in Dori (2011) and Crawley et al. (2016).

The configuration of a system is, therefore, comparable to the definition of software engineering in that it contains an arrangement of components and interconnection between those components as stipulated in IEEE (1990). To this end, runtime execution defines by the period within which the system executes, IEEE (1990). Therefore the runtime configuration of a system proposes the arrangement of components during a systems execution phase. Effectively, the arrangement of the system during operation.

Concerning definitive views on configuration, configuration management’s primary goals underpin the importance of the structural composition of a system, change tracking control and reporting, and product completeness and consistency, shown in Dart (1991a). Additionally, the functional capabilities that support ongoing system change through dynamic behaviours see configuration as the underpinning ecosystem to manage an ever-changing system. Capabilities such as dynamic and static configuration binding, hierarchical compositions, and simulation bound configurations seen in Dart (1991b), positions the need for a

structured understanding of the system under specification. The extension of the configuration into vertical models are evident in literature such as Burgess (1993), TC-SMG (1996), IEEE (2000), Burgess and Mende (2005), and likewise into theoretical states such as Petrie (2012) and Reichert and Weber (2012). The generalised view on configuration produces configuration baselines and process execution orders as shown in Petrie (2012) and Reichert and Weber (2012). A more IoT specific view of configuration is handled in Perera et al. (2014) and Ashraf et al. (2016).

The existence of preconditions enable parameter variation that supports adaptation and self-* property requirements such as self-configuring, self-healing, self-optimising and self-protecting, as covered in Subramanian and Katz (2000), Cheng (2008), Anthony et al. (2008), Swanson et al. (2014), Cirani et al. (2014), and Villegas et al. (2017). Specific evidence related to failure avoidance using an understanding of configuration is seen in both Cheng (2008), and Swanson et al. (2014). The precondition criteria for failure avoidance could be summed into the existence of a failure or forecasting of a failure before the failure avoidance process initiates. A key systems viewpoint indicates that an object can exist without a process. However, a process can not exist without an object due to the process playing a fundamental systems transformation role in changing object states, ISO/TC 184/SC 5 (E) (2014)(pp.13).

A specific view on preconditions is indicated in the text by Jacobson (2004), in that differences exist between configuration, reconfiguration, design-time configuration, runtime configuration, remote configuration, and in-system configuration. Additionally, the abstraction of hardware-based configuration appears in Jacobson (2004). These abstracted access mechanisms are discussed in the interfacing theory in IoT devices, as seen in the prior section on IoT CPS literature. IoT system architecture and IoT runtime configuration are evident in recent studies. However, there is no conclusive body of research on how different systems architecture decisions on runtime configurations impact state handling in the IoT-Ce.

A deeper understanding of runtime configuration processes offers a viable means to provide greater flexibility in systems, as indicated in Fazio et al. (2013). Most self-* properties assume the existence of a configurable object or process before adaptation can occur. By extension, current IoT system architectures fall short of applying a suitable modelling standard that reduces complexity. By considering the runtime configuration of system components as part of a standards-based system modelling process, it offers a means to provide flexibility in the systems due to an understanding of the system's behaviour given pre-existing

configuration sets. Therefore, the configuration provides an understanding of the system's arrangement before, during, and after system deployment, effectively tracking configuration changes over time.

1.6.4 Architecture Configuration Decisions

In a view on systems and architecture, ISO/IEC/IEEE (2011)(pp.19), two philosophies exist that see architecture as a systems' conception that is subjective and representative of the properties of a described system. ISO/IEC/IEEE (2011) cites four metaphors for describing architecture;

- as a blueprint,
- as literature,
- as a language, and
- as decisions.

Primary architecture formulation, above other system formulation questions, guides the structure of decisions in the architecture. Although the primary contribution in the ISARC aims to argue for the applicability of OPM and systems thinking, the continuous decisioning relating to the impacts of different configuration options lays the groundwork to structure repeatable decisions for the ISARC. In formulating decisions for architecture, three essential considerations form the basis of the architectural constraints in a system. Crawley et al. (2016)(pp.326-327) indicate these as;

- defining the architectural space within which decisions exist,
- defining the sensitivity of the impact of architectural decisions such that it moves the result of the decision in orders of magnitude, and
- defining decisions that impact the architecture specifically.

To this end, the definition of a multi-dimensional architectural space such as the IoT-Ce is shown to contain combinations of protocols, device options, sensing options, and innate device system value. As shown in the section on the IoT body of work, parameters such as networking, communication, energy management, configuration, interfaces, identification, discovery, processing, interoperability, service enablement, user adaptation, and a plethora of selection criteria drive increases in combinatorial structure in IoT systems architecture. Thus, the natural growth of architectural options in the IoT-Ce has shown to provide coverage instead of interactivity within the architectural space. Hence, the IoT body of work

is fundamentally coverage-based and not necessarily a true view of the systems architecture of a complex system. Architecture decisioning and representation of compact structures are primary design considerations in complex systems; Crawley et al. (2016), and Eppinger and Browning (2012).

The codification of architectural decisions into an electronic form solidifies the understanding of the architecture and arranges a seemingly abstract visualisation. Crawley et al. (2016)(pp.332) defines the decisions in system architecting as being non-routine, weakly defined, ordinarily having a significant impact, and solved by heuristic search or general problem-solving methods based on imprecise system models. The term manifests as “non-programmed” decisions by Crawley. Additionally, the view on human decision-making is that it is fundamentally flawed. Bahill and Madni (2016)(pp.159) indicates that there exists a need to distinguish between prescriptive and descriptive decision making. Where rational prescriptive decisions require a goal, perfect information, and a course of action that maximizes the decision’s output, individuals tend to make decisions descriptively. A means to sufficiently state the boundary, introduce decision information that maximally defines the impact of the decision, and remove unnecessary information is grounded in systems thinking. Furthermore, these architecture principles appear via the decision heuristics seen in Crawley et al. (2016)(pp.326-327).

Applied architecture variations contain a series of parameter choices. A configuration, as previously stated, contains the arrangement of variables within the systems architecture. In IoT systems and architecture, models that reduce the architecture to a series of sensitivity-bound parameters can be seen as state of the art in the IoT body of work. Methods such as the Architecture Trade-off Analysis Method (ATAM) contain attribute sensitivity outputs, Washizaki and Tei (2019) and Kazman et al. (1998). The ATAM enables the creation of attributes that apply trade-off and sensitivity parameters between different decisions. Additionally, complexity reduction forms such as Design Structure Matrices (DSM) lend themselves to programmatic optimisation mechanisms. The DSM enables a compact structured view of interactions between objects, processes, and states, as seen in Crawley et al. (2016) and Eppinger and Browning (2012). Combining these models proposes a decision method, which is largely a convergence between two established architecture tools. A suitably generalised use case is, however, necessary to prove the fit of the proposed decision method for the IoT-Ce. The selected use case drives a set of concrete problems against which decisions in the ISARC tests with high fidelity and credibility. By proposing the use of a PI control model, the decision method can be tested for error correction principles based

on instantaneous and accumulated errors. Therefore, the primary purpose of the method is to support configuration decisions where the communication of state is a beneficiary value requirement.

Formal validation of a broader architectural decision model that supports the modelling of mass IoT-Ce device instances could serve as a reference from where possible deployment paths could support additional general IoT use cases. Considering that more IoT edge control will move closer to the end-user, an understanding of the integration of packaged mass-market sensors and actuators will continue to play an essential role in delivering essential IoT services. In response to this need, sound architecture decisions propose a partial deterministic system behaviour while the system is in operation, namely at runtime. As interconnected systems expand in coverage, modelling approaches with greater linearity that cut across systems boundaries are proposed to solve this problem.

1.6.5 A Converged Literature Summary

Figure 1.6 presents the primary reference summary. The primary reference base in Figure 1.6 suggests the use of a limited set of standards within the IoT domain, even though a substantial standards reference base is already known for IoT, Postscapes (2017). These more specific protocols are considered secondary references in the study’s exploratory phase. They could become part of the system or impact the system from outside the defined system boundary. Systemic and environmental boundary placement is core to ISO/TC 184/SC 5 (E) (2014). Additionally, where state management is discussed as a supporting factor in current research, the literature presented below binds discussions about state directly with systems architecture requirements. This viewpoint exists based on the importance of state in OPM design. Figure 1.6 outlines a repository of each of the bounded classifications in the research library.

The convergence of the literature suggests that model validation is evident. However,

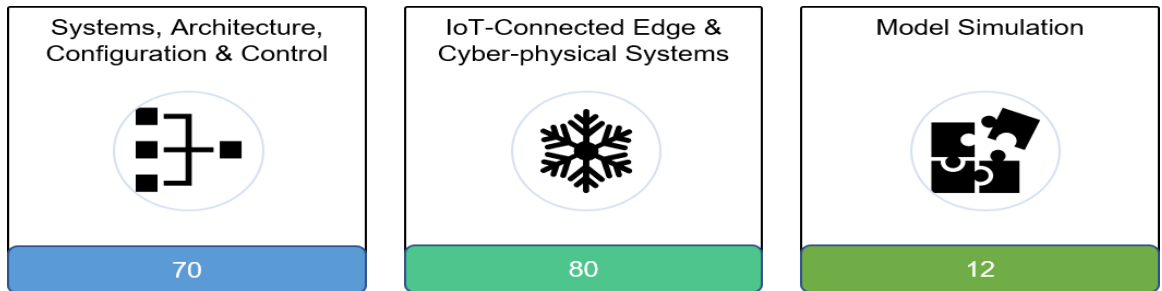


Figure 1.6: Research Topic Reference Count

most modelling efforts in the literature summary focus on simulation instead of structural and behavioural decision testing for CPS and less so for IoT. The body of existing knowledge for IoT leads to a view that system modularity can occur for high-level system concepts. Additionally, the existent primary purpose for the IoT is focussed on self-* properties to ensure seamless interaction. Therefore, an understanding of what underpins this purpose is the primary objective of the study at hand. Where self-configuration is a primary process, there has to exist an operand that undergoes change through the execution of a configuration process. As shown in the systems architecture literature in ISO/TC 184/SC 5 (E) (2014), a primary process plus operand equates to the function of the system, which supports the purpose of the system, which satisfies a beneficiary need, and so forth. The literature summary outlines the ability to structure the system and the boundary of the system that is under consideration at a conceptual level. A large opportunity exists to use the OPM ISO-19450 standard to model the system at differing granularity levels while maintaining form-function mapping. Table 1.1, proposes the classification of the reference list into the bounded research themes.

A Systems Architecture for IoT-Connected Edge Runtime Configuration (IS-ARC) decomposes to a suitable fit for handling a cross-domain thesis. Where systems architecture decomposes into systems and their fit into general systems design, the ISARC's architecture component enables arguments on the structure and behaviour mapping of the components in element level analysis. Thus, the view on systems architecture is undoubtedly crucial in the abstraction of interacting elements in a complex system such as the IoT-Ce. The IoT-Ce, which CPS underpins, contains a broad population of heterogeneous devices and purposeful use cases that establishes a suitably bound system that argues for in-system and out-of-system boundary placement. Runtime configuration argues pre-existence criteria in the form of ordered and sequential understanding of parameters that are important to the IoT-Ce's functioning. The references in Table 1.1 binds the argument for convergence of generally accepted systems viewpoints to a well-bound vertical domain - the Internet of Things Connected-edge.

1.7 Scope & Constraints

The ISARC applies a scope and constraint set using Dori's systems formulation criteria. In OPM, the modelling of a complete system is driven by five questions, as seen in Dori (2019b). The determinants of ISARC scope and constraints are;

Table 1.1: Master Reference List

Systems, Architecture, Configuration & Control contains 70 references
IEEE (1990), Dart (1991a), Dart (1991b), Burgess (1993), Karamanolis and Magee (1995), TC-SMG (1996), Skyttner (1996), Feiler and Li (1998), Kazman et al. (1998), IEEE (2000), Maier and Reichtin (2009), Subramanian and Katz (2000), Dori (2002), Marriott (2002), Soderborg et al. (2003), Dori et al. (2003), Jacobson (2004), Wagner and Wolstenholme (2004a), Robinson (2004), Wagner and Wolstenholme (2004b), Burgess and Mende (2005), IBM (2005), Thiele and Wandeler (2005), Rumppler and Elmenreich (2006), Feiler and Hansson (2007), Anthony et al. (2008), Cheng (2008), Gobshteyn and Dori (2008), Leon-Garcia (2008), Touretzky (2009), Lundberg (2018), Domer et al. (2009), Høyland and Rausand (2009), Rammig et al. (2009), Rao et al. (2009), Aubrun et al. (2010), Dori (2011), ISO/IEC/IEEE (2011), Bradshaw et al. (2012), Sharon et al. (2012), Eppinger and Browning (2012), Petrie (2012), Reichert and Weber (2012), Janert (2013), Mordecai and Dori (2014), ISO/TC 184/SC 5 (E) (2014), Ptolemaeus (2014), Rychly (2014), Smolin (2014), Swanson et al. (2014), Khaitan and McCalley (2015), Bahill and Madni (2016), Crawley et al. (2016), Dori (2016b), Dori (2016a), Brooks and Lee (2017), Mordecai and Dori (2017b), Al-Fedaghi and Alahmad (2017), Villegas et al. (2017), Cameron et al. (2017), OMG (2017), Incose (2018), de Matos (2018), Lee et al. (2019), Dori (2019b), Dori (2019a), Dori et al. (2019), Li et al. (2019), DSMWeb (2019), Sobol and Dori (2020)
Internet of Things & Cyber Physical Systems contains 80 references
Eman et al. (2002), Henzinger and Sifakis (2007), EPCGlobal Inc. (2008), Kopetz (2008), Vermesan et al. (2009), Marwedel (2010), Ailisto et al. (2011), Kopetz (2011), Uckelmann et al. (2011), Vermesan et al. (2011), Laplante and Ovaska (2012), Lee et al. (2012), Seshia and Lee (2012), Miorandi et al. (2012), Bassi et al. (2013), Fazio et al. (2013), European Commission CORDIS (2013), Khan et al. (2013), Sridhar (2013), Pei et al. (2013), Qiu et al. (2013), Ahmad (2014), Cantelon et al. (2014), Chuah (2014), Cirani et al. (2014), Anderson and Rainie (2014), Bormann et al. (2014), Marwedel (2014), Olsson (2014), Perera et al. (2014), Stankovic (2014), McLeod (2014), Yong-Fei and Li-Qin (2014), Behera et al. (2015), Datta et al. (2015), van Zyl (2015), GSMA (2015), IEEE (2015), Latronico et al. (2015), Lohstroh and Lee (2015), Taleb et al. (2015), Trifa et al. (2015), Ashraf et al. (2016), Barnaghi and Sheth (2016), Bera et al. (2016), Eclipse (2016), Shelby et al. (2016), Lee et al. (2016), Horn et al. (2016), Android (2017), Kostic (2017), Rubenstein (2017), Schoitsch and Niehaus (2017), Mordecai and Dori (2017a), Gartner (2017), IBM (2017), IDC (2017), Instructables.com (2017), Kaur and Sood (2017), Khazbak et al. (2017), Liu et al. (2017), Postscapes (2017), Sinche et al. (2017), Yu et al. (2017), Zhang et al. (2017), Zheng et al. (2017), Shaikh et al. (2017), Brooks et al. (2018), Jain and Mohapatra (2018), Arrow.com (2018), GSMArena (2018), IDC (2018), Linux (2018), Varga et al. (2018), Chang et al. (2019), Kharpal (2019), EvaTech (2019), IoTOne (2019), Kellogg (2019), Washizaki and Tei (2019)
Model Simulation contains 12 references
Pang et al. (2004), Oliphant (2006), Kondo et al. (2010), Sargent (2010), Yigitbasi et al. (2010), Epema et al. (2014), Haapala (2014), DeNero et al. (2020), Adhikari et al. (2018), SciPyCommunity (2019), Plotly Technologies Inc. (2020), Adhikari et al. (2020)

1. **Purpose**, which describes the system's reason for existence. It consists of a group or set of objects exhibiting primary attributes containing states and a main process that transforms an attribute between different states,
2. **Function**, that describes the main beneficiary value realised from the system. For example, in the case of modelling a social system such as a Conference, the main process ([Conference Occurring](#)) transforms business cooperation from loose to tight, Dori (2019b). In OPM, color scheme usage identifies differentiation between objects, state, and process. Objects and state occur in [green](#) when modelled on an OPM canvas, while processes occur in [blue](#). A function occurs as having a main [process](#) that acts upon an operand, or main [object](#), of the system. One of the key architecture analysis guidances is the use of gerund verbs in the OPM ISO-19450 standard for

process naming (section B.6.3) ISO/TC 184/SC 5 (E) (2014),

3. **Enablers** such as an agent (human) or instrument (non-human) that interacts with the system,
4. **Environment** such as the external environment that impacts the system under control, and
5. **Problem Occurrence**, that describes the reason for the system's need. In the case of the Conference, a decline in business ([Business Declining](#)) could be the reason why the Conference system is needed.

A systems' viewpoint for scope occurs through natural boundary discussions in the system. Specifically, the boundary draws between environmental versus systemic objects and processes. Objects outside of the control of the primary systems architecture realise as point 4 in the system's formulation criteria list above. Thus, the boundary that forms is the IoT-Ce and not the IoT ecosystem in its entirety. Additionally, the non-functional requirement set that validates against the dependability data set does not stretch towards all non-functional requirements but rather reliability only. Furthermore, within the IoT-Ce are various subsystem components that support different IoT use cases such as healthcare, home automation, and personal security. However, the core scope of study within the IoT-Ce is the growing mobile smartphone population and the applicability of dealing with mechanisms to access onboard device state.

A conceptual model that adheres to the OPM ISO-19450 standard is yet to appear in IoT-Ce research. As far as reasonably known, OPM ISO-19450 applies predominantly in use cases in the aerospace industry, de Matos (2018) and Li et al. (2019). Additionally, OPM applies in the analysis of cyber-physical state perception issues termed CPGAME - the Cyber-Physical Gap Analysis Method, in Mordecai and Dori (2017a). In Mordecai and Dori (2017a), the authors produce a post mortem analysis to propose a mitigation of the disastrous impacts of the 1979 Three Mile Island Unit 2 nuclear accident. This work indicates the handling of state misalignment in critical systems. Recently, interest has been shown in requirements and protocol engineering, Mordecai and Dori (2017b) and Mordecai and Dori (2014). However, application in generalised IoT systems appears to be less evident. The requirements for critical system design in Industrial IoT (IIoT) are certainly a future research area due to the need for hard real-time constraints similar to those in the existing OPM body of work. Consequently, the scope in the ISARC introduces a single domain in IoT as the primary beneficiary provider in the system architecture. A primary beneficiary ensures

that the system's value can be measured according to the value delivered by the provider system itself - An IoT Connected-edge system housing mobile smartphone instruments.

The systems architecture boundary uses the OPM ISO-19450 modelling standard to;

1. apply a model-based systems engineering (MBSE) approach using OPM ISO-19450 as a symbolic conformance model for the IoT-Ce,
2. enable model validation for an IoT-Ce device use case,
3. argue the requirement to understand pre-existing runtime arrangement of the system, in order to support self-* properties,
4. create a decision trade-off model that houses interaction decisions by proposing a method for ongoing configuration decisions for the IoT-Ce.

The flexibility within the OPM standard enables conformance at different levels. Where the ISARC scopes the first conformance level - a partial conformance - it still provides ample availability of symbols and elements such as symbolic link usage. An OPM Full and Toolmakers conformance level apply the remainder of the standard's path execution semantics and OPL generation requirements, respectively. Due to the fixed output for conformance, the scope of partial conformance requirements is sufficient to represent the ISARC. A partial conformance level is indicated in ISO/TC 184/SC 5 (E) (2014)(pp.10) as containing;

1. symbolic conformance to objects, processes, and states,
2. symbolic conformance to OPM syntax and semantics in cases of procedural link usage,
3. symbolic conformance to all other link types such as transformation links, consumption links, enabling links, control links, and condition links,
4. symbolic conformance to structural links and cardinalities and
5. symbolic conformance to logical operators.

In scoping the case study selection for the additional thesis consideration - decisions supporting dependability through state awareness - the case study is carefully considered to produce a generalised position on handling ongoing decisions for state awareness within an IoT system. These decisions produce a key requirement for optimising data transmission to and from the core smartphone population, henceforth termed IoT Edge Devices (EDs). This requirement is needed to produce efficiencies in the communication path between devices

that play a role in IoT efficiency use cases, such as those argued by Rubenstein (2017) and Kaur and Sood (2017). Similar to the layered view of the architecture for energy efficiency, in current research for edge computing, a layered architecture is presented to decouple the core cloud serving systems and those closer to an IoT-Ce user. As indicated in Yu et al. (2017), Jain and Mohapatra (2018) and Chang et al. (2019), mobile devices serve as edge components in both cloud and edge architectures. Three conceptual models indicate possible scoping of IoT study. These are indicated in Figure 1.7, 1.8 and 1.9.

In Figure 1.7, smart EDs communicate directly with each other, without traversing network

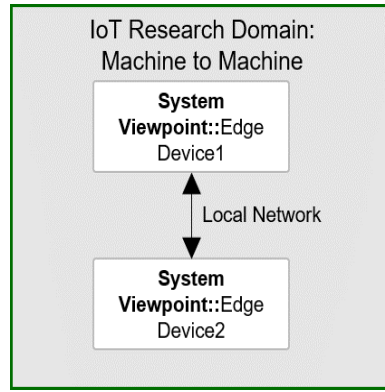


Figure 1.7: Machine to Machine Research Domain adapted from Yu et al. (2017), Jain and Mohapatra (2018) and Chang et al. (2019)

operator system boundaries. Examples of this are intelligent edge devices communicating directly with each other to produce an emergent function. For example, a sensor can trigger the activation of an actuator in a location other than where the sensor exists without traversing a mobile or local user network. In Figure 1.8, smart EDs communicate directly with a

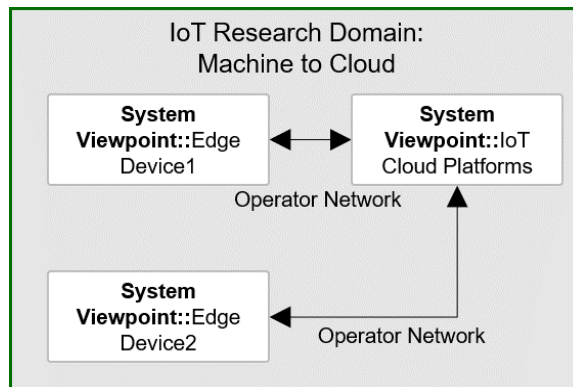


Figure 1.8: Machine to Cloud Research Domain adapted from Yu et al. (2017), Jain and Mohapatra (2018) and Chang et al. (2019)

cloud provider by traversing a network operator's system boundaries. Examples of this are

intelligent EDs using an operator’s network to produce an emergent function. For example, a sensor can upload an event to trigger an alert function on a different user’s system, such as a low battery threshold triggering an automated notification process to another system or subsystem that requires this notification. In Figure 1.9, smart EDs communicate via a

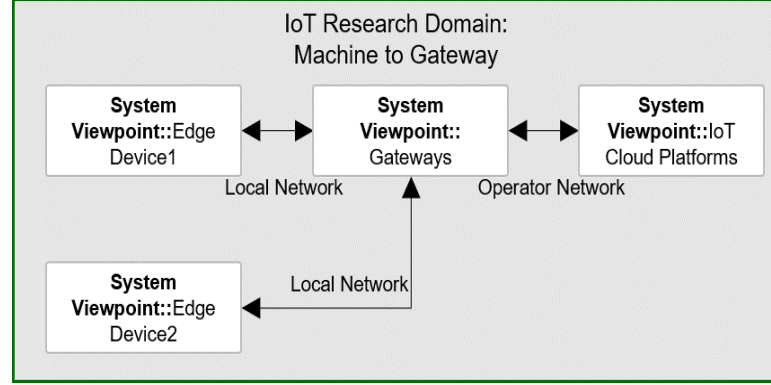


Figure 1.9: Machine to Gateway Research Domain adapted from Yu et al. (2017), Jain and Mohapatra (2018) and Chang et al. (2019)

gateway with a cloud provider by traversing both local and a network operator’s system boundaries. Examples of this would be intelligent home EDs tethered to a mobile device or router to produce an emergent function such as [Device Information Sharing](#). For example, the sensor can fall under the control of a mobile or fixed gateway to consolidate communication towards the cloud provider. A conceptual position setting a strict research scope boundary can therefore be represented by aggregating the three discrete system boundaries, represented in Figure 1.10.

The scoping that applies to a reliable communication use case that argues for increased

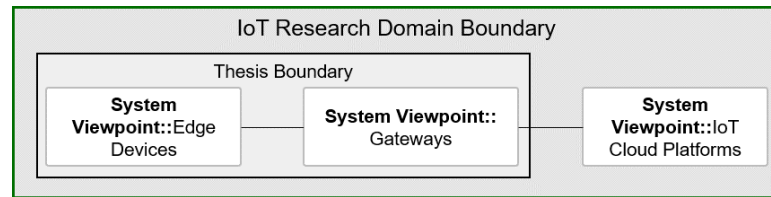


Figure 1.10: ISARC System Scope Overview

state awareness presents a finite set of ED state changes that aim to prove connection establishment between an IoT application and a DNS architecture. DNS architectures are known to provide a lightweight hierarchical structure for local and remote hostname resolution. As indicated in the characteristics that align IoT to embedded systems, reusable populations (ED/GW capable mobile smartphones) and protocols (DNS) enable a higher degree of sys-

tem value by meeting the core IoT requirements such as identification and communication. The use of conventional resolvable addressing protocols in DNS supports Object Naming Services for ED identification, EPCGlobal Inc. (2008). Additionally, the reuse of internet protocol-based interaction standards such as web services communication is also a known IoT standard, Bormann et al. (2014). This view is primarily backed by IoT interfacing theory indicated in Latronico et al. (2015), and Lohstroh and Lee (2015). Given that both a connected IoT ED state and the payload sizes are limited in resource budgets, the optimisation of communication and efficient transmission becomes an essential requirement for creating a state-aware IoT-Ce. Defining the study scope and constraints within the context of communication dependability in a closed system is thus required. This bounded system viewpoint produces the scope of the thesis in Figure 1.10.

Considering constraints, and due to a vast and often shifting body of knowledge for IoT, the constraints of the system occur within the agreed boundary outlined in Figure 1.10. One of the state of the art approaches that supports the modelling of constraints is founded in the ability to not only outline system realisation (Purpose, Function, Enabler, Environment and Problem Occurrence in Dori (2019b)) but also to reduce complexity. Reducing complexity is embedded in the view that systems are composed of hierarchies, Dori (2002)(pp.385-386). The strategy of stating hierarchies enables the standardisation of objects, processes, and states using a simplification and a standards-based model validation strategy. Furthermore, the modelling of objects, processes, and states that control the system versus those that are environmental serves as an exciting system constraint problem. Specifically, where large populations of heterogeneous components interact to provide different value to system beneficiaries. The systems architecture approach underpinning the ISARC is an OPM parameter extraction filter that delivers constraints in the object/process pre-existence requirement in OPM.

A set of constraints for setting the IoT-Ce system boundary produces a preliminary viewpoint based on what IoT is and what it is not. In their view on IoT as a domain, Vermesan et al. (2011)(pp.17) outlines a series of characteristics on what IoT is not. It is not;

1. pervasiveness and ubiquity as these do not imply the use of objects. This view can be supported by the pre-existence of objects and processes seen in ISO/TC 184/SC 5 (E) (2014)(pp.13),
2. the internet protocol (IP) as many IoT devices might not require internet connectivity, as suggested in Figure 1.7,

3. communication technologies, although it does play a part. This viewpoint references communication as an enabler instead of the primary purpose in driving IoT-Ce system value,
4. applications, and services used to describe the IoT. This description is akin to describing applications and then using those descriptions to describe the internet. Here the viewpoint references applications and services as enablers instead of the primary purpose in driving IoT-Ce system value.

The authors in Vermesan et al. (2011) view these characteristics as “enablers” to implement the concept of the IoT in different contexts. In OPM system terms, these enablers are instruments (non-human).

Constraint Set

1. The IoT-Ce system is regarded as *closed* once an IoT device becomes attached to a local or operator network. The ability of the EG/GW capable mobile smartphone, termed a Front-end System Set, to initiate communication under a constraint that all required components allow this communication. For example, firewall port configurations, device address advertising, and general availability. The notion of harder subsystem boundaries for edge components is noted as user-controlled front-end and near-end system sets and operator-controlled far-end system sets (Cloud), Yu et al. (2017).
2. Front-end system set instances *increment independently in a non-sequential manner* from a single node towards the physical or logical upper limits of the DNS systems hierarchy. Local and remote DNS components implement a variable technical limit in terms of the number of hosts (Front-end system sets) it can respond to in parallel,
3. Front-end system set instances are *enrolled automatically* using gateway initiated interactions. Partial and Manual enrolment is not explored in detail; however, for brevity, detail is included in Appendix B,
4. internal Front-end system set processing delays have a *variable upper limit*.
5. Front-end system set instances can enter into several *sleep and standby states* in between transmission, Rubenstein (2017). Thus, an optimisation strategy concerning scheduling of application processes is considered such as those indicated in Yu et al. (2017),

6. the request for Front-end system set states from a requesting IoT application could *traverse a combination of network protocols* via distributed gateways and control subsystems, Yu et al. (2017). Yu et al. (2017) indicates that the shortest distance will reduce transmission time. To reach a target Front-end system set, increases in request-response round-trip time is *variable*, and
7. the processing of data for model validation purposes is limited to *simulated control*. It could include post-event analysis of events supporting the optimisation of control decisions. A standard Proportional and Integral (PI) control strategy applies to the simulated control proof.

Constraint Set for an OPM Control Diagram Output (OPD and OPL). Adapted from Janert (2013)

1. control initiation measurement, measurement time period, target control position and time step are *variable*,
2. a set of standard control gains are *applied periodically on the time step*,
3. requests from the Front-end System Set not handled are treated as *failed state-notification requests*,
4. *no queued requests* to be processed during or after the measurement time period,
5. *a finite number of request and responses* from the Front-end System Set and
6. *a random load intensity* over the measurement time period.

1.8 Approach & Limitations

The suggestion that OPM can support the creation of the ISARC creates an exploratory position for the systems architecture then outlines possible causal relationships between architecture variations and state-awareness for the IoT-Ce. Therefore, the approach models; IoT-Ce system interactions based on the IoT body of work applied to a set of abstracted Front-end System Set interfaces, and real-world process interactions for runtime execution modes of Front-end System Set instances. Therefore, a systems approach is necessary to ensure a rich structured understanding of the study's context in question with suitable abstraction levels. Namely, which system configuration decisions can possibly impact IoT-Ce communication dependability. With varying levels of maturity seen in the vertical research

domains, the approach should outline the extent to which a convergence exists between Systems, IoT, and Configuration domains. Understanding that each domain has a unique history allows the approach to position the limitations that impact vertical and cross-functional domains. Figure 1.11 presents the approach concepts per the thesis coverage.

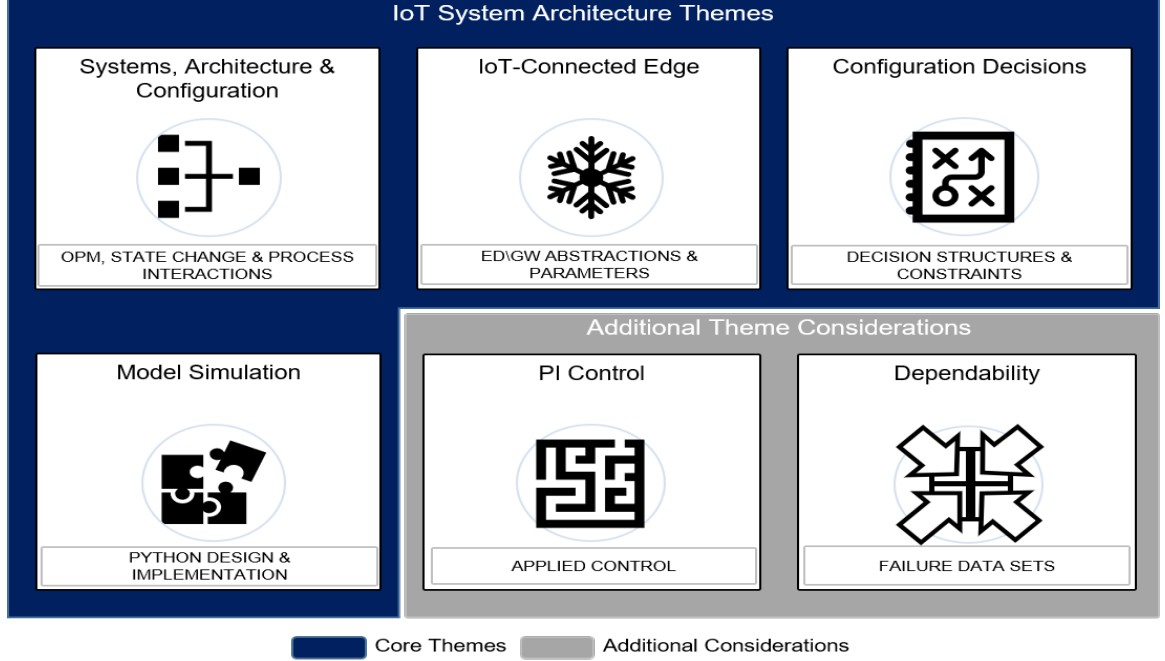


Figure 1.11: Approach and Limitation Overview

Systems Architecture: In approaching the study as a structured architecture design process, the proposed approach considers the purpose of the system as a primary design attribute. Indicated in sound systems design using model-based principles in Dori (2019b), the purpose of the system describes the system's reason for existence. The purpose of the eventual system is represented by an architecture that supports different parameters for making decisions relating to the system components that support efficient state-awareness. The systems architecture contains the structural and behavioural aspects of the specified system as per Dori. However, a process to deliver a system can also apply systems thinking such as SIMILAR in Bahill and Madni (2016). The approach of modelling a system is a structured view on identifying the problem space, identifying alternatives in the architecture, modelling the system, and integrating the system, Bahill and Madni (2016). Bahill further indicates the approach to contain the deployment of the system, assessing performance, and re-evaluating the system in an iterative decision process. The approach for delivering the ISARC proposes three separate concerns;

1. using the core OPM conformance requirements to model the ISARC by answering key

system delivery questions related to *purpose, primary function, enablers, environment, and problem occurrence* as seen in Dori (2019b),

2. maintaining hardware-software model co-design considerations as indicated in the CPS body of knowledge which supports Front-end System Set access abstraction as seen in Lohstroh and Lee (2015) and Latronico et al. (2015), and further underpinned by Domer et al. (2009), Marwedel (2010) and Marwedel (2014), and
3. using a structured decision process to deliver parameter-based decision outputs as seen in Bahill and Madni (2016), Kazman et al. (1998) and Washizaki and Tei (2019).

Three approaches to OPM ISO-19450 conformance exists, ISO/TC 184/SC 5 (E) (2014). These are; conformance at a symbolic level, full conformance, and conformance for tool-makers. A limitation of using full conformance of the ISO-19450 standard is to exhibit many objects and processes, both environmental and systemic, that could yield full system function. However, the guidance provided for modelling with OPM is to consider model completeness versus clarity continually, Dori (2011). As noted in the hypothesis, a low-level characteristic such as a control parameter set should be sufficient to support the ongoing abstraction of other dependability characteristics in a system. This support will yield a set of parameters against which ongoing decisions relating to the arrangement of system components are made. The limitations of using OPM are only limited to identifying systemic and environmental components in a system as broad as IoT. By proving the use of OPM for the IoT-Ce, it reasons that a natural reduction in complexity should be sufficient to extract requirements from a body of knowledge and present these as architecture decisions to aid in IoT-Ce systems architecture design. The core reasoning behind the use of PI control is the concrete nature of problems that sets it apart from other modelling problems. In modelling the systems architecture, the asserted properties about the system are, in fact, properties of the model itself, as noted in the Kopetz principle, Ptolemaeus (2014). Thus, the approach and limitation for assessing OPM as the primary systems architecture method for the IoT-Ce in this thesis is in;

1. the ability to structure individual parameter outputs that represent real world system constructs using a partial OPM conformance strategy,
2. the ability to create a suitable boundary that supports IoT-Ce and runtime configuration arguments, and
3. generalising parameter choices extracted from selected IoT literature such that it informs system *purpose, primary function, enablers, environment, and problem occurrence*.

IoT-Connected Edge: The limitations of applying a method that fits systems on the IoT edge is that core environmental systems that impact IoT-Ce behaviour might not be covered entirely. For example, the coverage for other areas of IoT, such as IoT cloud, IoT analytics, IoT security, or any other specialised IoT domain, could result in a more generalisable decision model. However, the decision parameters would be highly volatile due to rapid increases in the growth of generalised IoT. In generalising a larger area of study, it would entail a broader structural and functional analysis against which environmental objects would be converted to systemic objects. Effectively, changing the view of environmental object impacts on the core IoT edge system, thereby increasing decision parameter variation and not exhibiting system boundary conditions in a controlled system set.

To minimise the impact of change for extended study results outside of the IoT-Ce, a smaller set of system interactions are generalised for the systems architecture. This reduction in the study area enables a smaller subset of interactions to be validated before assessing more general problem occurrences. An alternative approach could be to extend the systems domain reach; however, the risk of creating a model that defines a protocol-only model is highly likely. This view is not core to the hypothesis, and it risks producing a view contrary to what IoT is and what it is not. As indicated in the constraints by Vermesan et al. (2011), IoT is not merely the applications or communication technologies that represent system beneficiary value. The area of protocol specification using OPM is already a recent area of interest, Mordecai and Dori (2014).

Finally, IoT platform providers produce automated and manual device discovery and enrolment processes. As seen in Varga et al. (2018), the Amazon web service IoT platform houses capabilities for device registration, device inventory management, and policy enforcement. Additionally, this platform enables a service called “Device Shadowing” that enables virtual state management of connected Front-end System Set instances. Device discovery is discussed in Appendix B; however, the detail is not tested in the OPM configuration variation set. The limitation of this approach is that cloud system capabilities will not be included in the architectural decision model. This approach is preferred to establish the required front and near-end system boundaries and not richer cloud system capabilities, or their impact on devices in the IoT-Ce. Thus, the limitation for assessing mobile devices as the primary instruments in the IoT-Ce in this thesis is in;

1. the ability to view a generalisable mobile population that exposes abstracted interfaces,

2. the agreement that edge intelligence will increase in generally available smart mobile devices, and
3. increases in the autonomous control of smart mobile device applications.

Runtime Configuration: The study’s intended limitation is that configuration capabilities shall be modelled as purely environmental, having the primary processes consume outputs of a configuration management system. The proposed capabilities in Dart (1991a) that could have an output impact are; *structure*, *components* and *construction*. Other capabilities such as *auditing*, *accounting*, *controlling*, *process* and *team* is regarded as pure configuration management functions. The study does not aim to produce or defend the use of configuration management systems; however, advances in automated architecture decision making could challenge the view of understanding configuration before self-* process execution. Limitations in current literature are that there can only exist a finite number of configuration variations until artificial intelligence, machine learning, and deep learning becomes mainstream in IoT systems. Furthermore, the variation in protocol semantics makes the learning and application of these protocols fairly complex. IBM (2005) introduces a multi-stage structured approach for handling adaptive systems behaviour, namely, Monitoring, Analysis, Prediction, and Execution (MAPE). The application of MAPE-K knowledge management to support automated IoT architecture decisions is still uncharted territory. However, the application of MAPE-K to handle IoT control using finite state machines can be seen in Lee et al. (2019).

The approach to tackle runtime configuration is carefully considered as self-* properties require a degree of pre-existing knowledge of the system to enforce self-* actions. Therefore, the understanding that an object, process, or state is needed to underpin ongoing self-* action is a foregone conclusion. The impact of this conclusion is that systems require a configuration or variation thereof in order to operate. The efficiency of this operation could manifest as the preconditions of effective self-* actions and responses. An alternative approach to consider is to reverse engineer systems that exhibit self-* properties as a primary activity in the design. The structure that impacts different configurations is known for existing frameworks such as MAPE-K and RAINBOW. However, the primary goal of these frameworks is to meet control objectives for self-adaptation. To minimize the impact of pure reverse engineering, runtime configuration is deemed the arrangement of components at runtime enabled through structured design activities. The approach is to clarify the use of configuration, reconfiguration, adaptivity, and self-adaptation to position the use of primary processes that need to be enabled before self-adaptation can occur. Other autonomic

systems frameworks, instead of MAPE-K can be considered alternatives; however, the core pre-existence criteria will be maintained given the hierarchies in general systems theory. The view on hierarchical systems structure is cited in Dori (2002), in the chapter on general systems theory. A limitation of viewing hierarchy as the IoT-Ce’s structural basis is that the IoT-Ce is seen as a system of interconnecting parts. This is termed a hierarchical paradox in Skyttner (1996)(pp.39). The belief in systems is that a multi-levelled hierarchy exists from static structures, to simple dynamics, to self-regulating behaviours, and then finally to a system set that determines its end state via the understanding of “absolutes” that exhibit systemic behaviours as cited in Dori (2002) and Skyttner (1996). Thus, the limitation for assessing runtime configuration in this thesis is in;

1. the ability to view the arrangement of components and subsystems in the IoT-Ce as hierarchical in nature,
2. autonomic frameworks that expect the existence of a system that requires corrective control, and
3. viewing the handling of configuration management output as systemic while viewing configuration management processes as environmental in the IoT-Ce.

Architecture Configuration Decisions: The approach for considering IoT-Ce architectural decisions positions a total upper limit of configurations for IoT-Ce devices. An n-squared matrix (DSM) structure applies this upper limit thinking. However, it allows an increase in the number of rows and columns to increase combinatorial variations to support complex structure growth. However, the guidance provided for modelling with OPM is to consider model completeness versus clarity constantly, Dori (2011). This guidance considers how many objects, processes, and states appear on a single Object Process Diagram (OPD) before simplifying the OPD without losing model clarity. Therefore, the use of DSM can be a limitation if unbounded requirements increase the IoT-Ce architectural design scope. In order to minimise this impact, the enforcement of strict subsystem and component boundaries must occur. Although the primary DSM structure is enhanced by creating attribute-based decision points, it is also susceptible to vast attribute combinations. Attribute-based decisioning enables decision analysis as single non-impacting decisions as indicated in Washizaki and Tei (2019) and Kazman et al. (1998). Morphological matrices seen in Crawley et al. (2016) enable attribute-based decisions as well; however, the conditional logic found in the matrices serves as an unnecessary overlap to core DSM deliverables. Prioritisation of key dependability characteristics for either DSM or morphological matrices tool is a suitable hedging factor for this limitation. A single dependability characteristic such as reliability

binds the scope and the systems architecture into a single system boundary.

Jupyter Lab 1.2.6 with Python 3.7.4, and IPython 7.12.0 presents the implementation of the ISARC model validation. Python 3 Notebooks (.ipynb) produces the model's concrete implementation. A combination of Python's libraries enables a standard implementation of the DSM. The NumPy library detailed in Oliphant (2006) enables simple baseline functions such as Identity, Diagonal, Eye. These functions natively expose n-squared matrix structures that can be programmatically modified. A limitation of using this approach is that many custom functions will be needed to implement a similar function that presents a standard function in Python. Additionally, specialised immutable array elements will have to be explicitly coded in diagonal intersections in the standard n-squared construct. By using standard libraries, diagonal immutable array elements are possible in Python using the writeable property of the `numpy.ndarray.flags` attribute, seen in SciPyCommunity (2019). Immutability serves the purpose of locking DSM diagonal cells. To recall, DSM with input columns (IC) indicates that each row parameter provides inputs to the column's associated parameter. Intersections of the same row and column parameter represent neither an input nor output as the parameter does not communicate with itself. This intersection is present where a parameter represents a logical component. Further decomposition of the component could yield an input-output structure, but this is only required if the context for communication is to state the internal process of the component itself. This decomposition relays a subsystem or component decomposition problem, but more so a completeness and clarity problem in ISO-OPM 19450 terms. The DSM-IC is a crucial recommendation for modelling ISARC hierarchy constraints simulations in Python.

The simulated Python data set is offset against a historical data set to suggest the applicability of parameter-based decisions against a known set of dependability (reliability) measures. The system that generated the historical data set is a known hierarchical web system that handles address resolution. In Pang et al. (2004), the authors specifically produce Domain Name System (DNS) availability and non-availability indicators using load distribution and different deployment patterns as the basis for their study. A limitation of this approach is that the ISARC model comparison tests against a system that could be argued as having a multi-node path towards the local or remote DNS host resolution servers. The argument that counters this is that a master-slave resolution path will initiate along the path from a primary instrument - the Front-end System Set instance - towards the addressing server and accessible in a local network or public network. The primary goal of applying a communication outbound from the Front-end System Set instance is handled

internally within the Front-end System Set instead of requesting connection permission from the addressing server. If the primary goal is to produce an addressed Front-end System Set instance, then the selection of path towards the addressing server becomes systemic instead of environmental. As with the argument for the pre-existence of a configuration, the form and function produced by the ISARC assess primary system value. This value is in the form of the existence of an addressable Front-end System Set instance, the availability of an assignable address range, and the existence of address objects before the process of addressing can occur.

The factors that bind the approach for the decision model for this thesis are; DSM for structure and attribute analysis using Python. Thus, the limitation in applying DSM as the primary decision matrix for the IoT-Ce runtime configuration parameter set in this thesis is in;

1. the ability to extract a viable parameter set that aptly describes configuration decisions for automated DNS communication of Front-end System Set instances,
2. the ability to apply standard Python scripting libraries in order to simulate model validation, and
3. the ability to apply a series of accepted constraints and control against a historical failure data set.

The ISARC aims to position an approach to simplify system complexities by stipulating the function and form of an IoT edge system containing generally available mobile devices. A reduction in complexity is in line with an existing research body in model-based systems engineering using the OPM ISO-19450 standard. The limitations encountered in simplifying a model are bound to variations of model viewpoints used to model a system. As seen in methods that use the Unified Modelling Language (UML) or Systems Modelling Language (SysML), a multi-view model viewpoint exists, increasing the risk of model ambiguity. Furthermore, the translation of a model to its associated textual representation is a two-stepped approach in standard modelling methods. Namely, model first and then generate a model-to-text representation of the model. To minimise the two-stepped approach, OPM yields a dual-channel principle that implements a simultaneous model-to-text representation of the system.

The selected body of knowledge for this chapter is broad enough to represent a systems

architecture yet well bounded to support system boundary principles. Hence, a set of configuration options can be discussed using natural progression from the form and function of IoT to IoT-Ce to CPS and then directly to the impacted parameters in a runtime configuration.

1.9 Thesis Structure

The remainder of the thesis contains the following chapters;

- **Chapter 2: The theoretical basis and fundamentals of Systems Architecture, IoT-Ce, System Configuration, and Architecture Decisions.** This chapter outlines background work and demonstrates a broader appreciation for IoT-Ce as an area of study supported by the primary themes of interest for the hypothesis.
- **Chapter 3: Systems Architectures and IoT-Ce Configuration.** This chapter presents a body of work as an integrated view of the primary themes of interest. It provides a critical assessment concerning the thesis objectives. It contains a comprehensive analysis of the critical requirements for the proposed systems architecture (ISARC) by applying the OPM ISO-19450 standard.
- **Chapter 4: Modelling the ISARC: Systems, IoT-Ce and Configuration** This chapter contains a comprehensive systems model for the essential requirements for the ISARC using the OPM.
- **Chapter 5: Examples and Evidence: ISARC Model Validations** This chapter presents the model validation and evidence for the function-to-form mapping in the preceding chapters. It provides a critical assessment of the results, which includes state-notification validations generated for different architecture configurations. The validated systems architecture is presented using the interactions, parameters, and constraints discussed in the preceding chapters.
- **Chapter 6: Conclusion** This chapter provides concluding remarks as a summary of work covered in previous chapters. It restates the research contribution and indicates future work. The results overview finalises the findings and presents this as the completed study objectives.
- **Appendix A:** This section contains PI control theory and other considerations for the ongoing design of the ISARC. Appendix A is cited across individual chapters to ensure that the systems architecture and decisions relating to the architecture can be decoupled without binding concepts such as control into architecture decisions.

Control should be seen as one of the many strategies that can be used to make sound architectural decisions to support reliability requirements in the IoT-Ce.

- **Appendix B:** This section contains the detailed analysis of requirements extracted from the thesis body of literature using the OPM parameter extraction filter. These requirements represent additional system architecture decision parameters that form the basis of additional considerations for the main contribution.

1.10 Chapter Summary

Chapter 1 focuses on presenting the unique overview of respective vertical areas of concern. As indicated, four primary characteristics drive the enablement of a systems architecture for the IoT-Ce. These are;

1. **for simplification** in resolving human cognitive constraints as seen in Kopetz (2011), and enabling of mechanisms to cope with complexity using configuration, Ailisto et al. (2011). Additionally, the use of co-design between software and hardware to reduce inconsistencies is also a known juncture between IoT and embedded systems, Stankovic (2014).
2. **for greater usability** by focusing on a population that contains usable components such as mobile devices, Kopetz (2011), and those used in discovery and closed-loop communication, Stankovic (2014).
3. **for reuse** as indicated by Kopetz (2011) through the use of existing standards to meet the needs of IoT use cases. For example, the use of resolvable addressing protocols in Domain Name Systems supports Object Naming Services for edge-device identification, EPCGlobal Inc. (2008). Additionally, the reuse of internet protocol-based interaction standards such as web services communication using the Constrained Application Protocol (CoAP), Bormann et al. (2014), is also a known standard.
4. **for management and optimisation** as indicated by Kopetz (2011), and smart object interaction across dimensions such as state and policy awareness, Vermesan et al. (2009).

This thesis produces a systems architecture that outlines the systems, subsystems, and components employed at runtime in the IoT-Ce systems boundary. The study of the runtime properties of systems can provide a better understanding of how to model systems that are dependable and scalable and have the requisite properties to find application in large-scale

distributed systems. The primary benefit of the contribution of this study is the identification of well-defined functional system, subsystem, and component boundaries that could provide different classes of configuration and profiles for IoT edge systems.

The use of mobile sensor systems for IoT-Ce use cases is growing at a rapid pace. While the demand for these integrated embedded systems increases, devices become more compact, resulting in more significant limitations for onboard capabilities. While the physical forms of these devices decrease, demand for more complex functionality increases. Thus, variations of configuration increase heterogeneity that mainly causes increases in stating the actual purpose and value of EDs in the IoT-Ce. Therefore lighter and more robust systems architecture methods should support decreases in IoT system deployment complexity. A suitably and efficiently modelled system can exhibit higher degrees of effectiveness if used in the correct context. A systems approach reduces the complexity of stating a systems architecture for the IoT-Ce. This approach stems from the belief that low-level IoT-Ce subsystem characteristics that influence the state of the system can abstract to a conceptual IoT-Ce system architecture to model dependability through increased state awareness. Increases in state-awareness support better management by reducing the gap between the actual and perceived state of Front-end System Set instances within the IoT-Ce.

The outcomes of this research combine the ideas of systems architecture and runtime configuration to deliver a tool kit to classify architecture problems at the early stages of an IoT-Ce application design process. Therefore, the proposed research study does not aim to re-establish individual vertical areas of interest in embedded systems or cyber-physical systems. The study aims to succinctly position a verifiable hypothesis by creating a standards-based model for proposing decisions on runtime configurations for the IoT-Ce. The positioning statements of the thesis are;

1. for system architecture **simplification** and **reuse** purposes; OPM ISO-19450 structures the architecture model,
2. for systems architecture **usability**; a complex system scales down to a single domain that creates a single purpose system boundary, namely the mobile device population in the IoT-Ce,
3. for system architecture **management** and **optimisation**; the arguments to understand pre-existing runtime arrangement of the system, in order to support self-* properties,

4. for system architecture **management** and **optimisation**; a simulation of a decision trade-off model that houses interaction decisions by proposing a method for ongoing configuration decisions for the IoT-Ce.

Chapter 2

An IoT Systems Architecture

“Systems-level thinking is more than just being systematic; rather, it’s about the understanding that in the ebb and flow of life, nothing is stationary and everything is linked.” Guru Madhavan - Think like an Engineer

From large-scale industry to consumer electronics, from general retail to solving social and humanitarian issues, people are increasingly dependent on technology. IoT systems consist of a mix of technology classes that apply to real-world problems. Within each technology class, a set of decisions exist that ensures optimal and cost-effective system design. Like many systems with a heterogeneous composition, IoT Systems suffer from similar standardisation problems as those found in the general computer systems industry. IoT systems, being fragmented and variably standardised, provides some interesting departure points for stating what could eventually lead to a coherent architecture. However, even in abstract form, the systems architecture must effectively describe the structural and functional aspects of the described system. This systems approach is a mandatory requirement as architecture is defined as a function to form mapping, resulting in the emergent behaviour of the system, Crawley et al. (2016). Thus, the representation of the IoT-Ce systems architecture produces an explanation of the relational aspects of the system and the resultant functional behaviour of the system.

The representation of the IoT-Ce system constructs until now has been rudimentary. It explains the broad concepts of the things that represent the devices that exist in the IoT-Ce ecosystem boundary. In Chapter 1, several references reflect the functional components of the architecture, such as data acquisition, data transmission, and data analytics. These are termed functional as they each inherently contain a **process** and **object** transformation, such as data acquisition that, when transformed, produces data acquisition or simply *acquiring data*. This function contains the process of changing the object (data) from data that has

not been acquired data that is acquired. Similarly, the function, *data transmission*, or simply *transmitting data*; represents the process that changes the object data from not-transmitted to transmitted. Finally, the function, data analytics, or in OPM terms *analysing data*, represents the process of changing the object (data) from not-analysed to analysed. In these functional examples, the function requires a form to produce the emergent system function such as the ability to acquire data, transmit data or analyse data.

However, in systems architecture, form is described using sequence, location, proximity, arrangement, and connectivity, Crawley et al. (2016)(pp.31). These are all crucial structural information blocks that exist in the IoT-Ce system before a process can be mapped to produce a function. Therefore, the IoT-Ce system defines as a set of entities and relationships that produce a functionality greater than the sum of its parts, as underpinned by the definition of emerging systems value in Crawley et al. (2016). Additionally, the system is described as being human-made and may contain configurations that form the basis of control through several automated or manual human-centric entities, ISO/IEC/IEEE (2011).

2.1 An existing approach for supporting IoT Architectures

Widespread popularity for defining IoT research has been the subject of academic and commercial agendas in recent years. These agendas position IoT as an ecosystem that contains subsystems and components. Thus, specifying IoT architectures, multiple functional subsystems, component types, and interactions can exist. With advances in technology, mobile devices now contain onboard sensor technologies that position these mobile devices as primary objects that receive and send sensor measurement - effectively enabling mobile devices to act as IoT ED and GW - a **Front-end System Set**. The specification of these functional mobile components can, of course, be modelled conceptually or logically.

The ecosystems that result from primary edge architectures are present in both commercial and open source technologies. However, once classed into a mobile device, gateway, and cloud applications, most architectures define the same recurring patterns. The architectural options in Figure 2.1 presents a modular viewpoint. These viewpoints contain a separation of concerns across typical IoT subsystems such as devices, gateways, and cloud applications such as those seen in Eclipse (2016). In further support of this view, component hierarchies appear in Yu et al. (2017) and Jain and Mohapatra (2018). It, therefore, follows that a general and well-understood view exists for system hierarchies in IoT. A framework outline shown in Eclipse (2016) creates a structural placement of vertical functions for these system hierarchies. The frameworks show that;

1. singular capabilities exist in software stacks for constrained devices. These capabilities lean towards pure sensor and actuator functions.
2. multiple capabilities exist in software stacks for gateways. These capabilities lean towards aggregation and communication functions.
3. multiple capabilities exist in software stacks for IoT cloud platforms. These capabilities lean towards a higher degree of aggregated function.

An aggregated architecture options overview is presented in Figure 2.1. These bounded architectures produce a decomposition of the research models presented in Figure 1.7 and Figure 1.8. In the constrained device systems boundary in Figure 2.1, functions such as

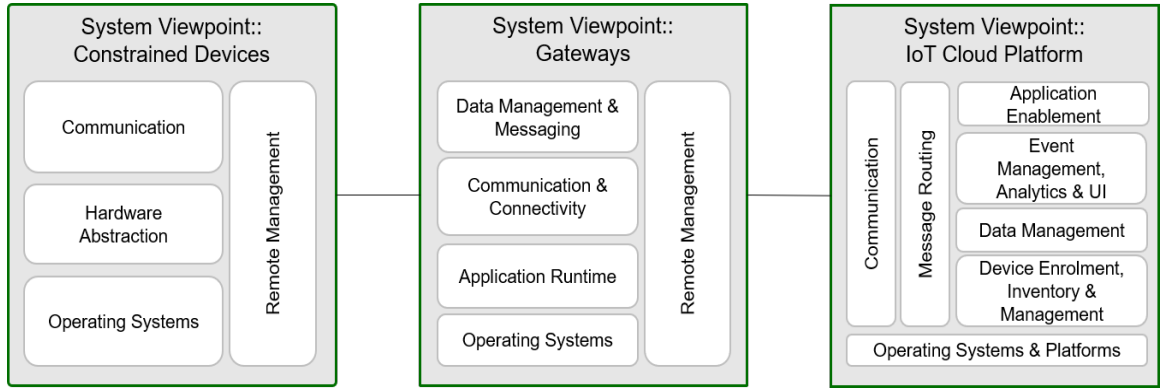


Figure 2.1: An Aggregated Architecture Options Overview adapted from Eclipse (2016)

communications, hardware abstraction, and operating systems appear evident in a single system boundary. Remote management is a shared function between the constrained device and gateway boundaries. Although the primary function appears shared, secondary and supporting functions will be different due to onboard resource comparisons between simple and more complex edge devices. The gateway systems boundary contains aggregation capabilities with the purpose of this system, quoted by Eclipse (2016), as a coordination function. As mobile smartphone devices do not necessarily require local network coordination, these mobile devices (**Front-end System Set**) can support direct connection to the subsequent significant processing IoT system node. The **Front-end System Set**, therefore, contains both constrained device and gateway functions within the same physical system boundary. This boundary expansion reduces the need for environmental functions to control the primary **Front-end System Set** functions. Eclipse (2016) indicate that these gateways execute server functions that support high-level programming languages such as Java, Python, and NodeJS.

An object, process, and state view similar to the domain functions mentioned for acquisition, transmission, and analytics can thus apply. Processes such as [Remote Management Handling](#), [Communicating](#), [Hardware Abstracting](#), and [System Operating](#), become the basis of the core system questions that produce problem occurrence, purpose, primary function, enabler, and environment. The extension further to the base questions is one relating to pre-existence in the system boundary. For example, what is the order of functions for the [Front-end System Set](#)? The [Remote Management Handling](#) process requires an object to exist to initiate a state change to the objects under the system boundary. The primary consideration for using the OPM thus benefits the IoT-Ce design process so that an ordered existence of objects appears in the system before a process can transform the state of said object.

2.2 Extracting Depth in Vertical Themes

The assessment of the vertical depth of each essential theme set - Systems Architecture, IoT, and Configuration - produces additional depth to the overall argument of why OPM is suited for the IoT-Ce. This assessment argues that the line between the current use of OPM, and the shortfalls of existing IoT-Ce architectures, can be drawn such that the essential principles of OPM design apply directly to the IoT-Ce. More so, OPM's use to stipulate what [Things](#) are, and what they are not, become second nature in IoT-Ce architectures as OPM inherently drives form and function-related questions.

Chapter 2 exists as a precursor to the integrated system architecture in Chapter 3. Chapter 2 aims to baseline the requirements ensuring that the compelling reasons relating to the core hypothesis are protected. At a minimum, it sets the hypothesis limitation that eventually governs the integrated system architecture boundaries en route to the concluding chapters.

2.2.1 Systems Architecture

2.2.1.1 A Systems approach towards Architecture for IoT

Structural systems and functional architecture have undergone substantial changes over time. In recent history, monolithic processing environments would span multiple buildings to more recent advancements where most user-required processing can execute on a single desktop or mobile phone. This evolution underpins the Internet of Things (IoT), which sees seemingly unrelated objects establishing communication and sharing data on a large scale, Bassi et al. (2013) and European Commission CORDIS (2013). Therefore, systems design requires considerations for understanding the impact of real-world constraints, which play

an essential role in the design, access, and management of devices found in single subsystems or multi-node interconnected systems. Thinking about the system as a whole made up of parts is the first point of departure and is core to systems thinking.

In conceptual representation, a system contains functions that work in tandem to produce the value promised by the system. The decomposition of the system into subsystems and components is vital to determine the functional boundaries of those entities in the architecture. In their view on decomposition, Crawley et al. (2016)(pp.310) indicate decomposition as the decision that provides the highest “leverage” in architectural design. The authors in Crawley et al. (2016) view the system’s decomposition into boundaries as an enabler of future systems architecture options that support effective trade-off and “explicit” design choices. Figure 2.2 represents three key areas of decomposition that provide a natural structural hierarchy within which the implementation of system components bind. A system is de-

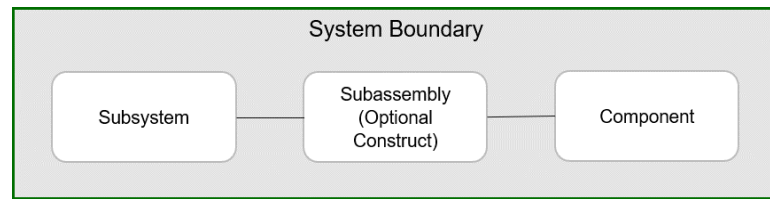


Figure 2.2: A Proposed Sensing Configuration Boundary. A Systems Architecture Decomposition Viewpoint

defined by the architectural subsystems or components configured to meet a specific purpose, operating in a specified environment, ISO/IEC/IEEE (2011). Hardware, software, data, people, processes, and procedures that make up a system broadly define the subsystems and components. Systems thinking is thus a process of thinking about a system as a complex interconnection of things that work in harmony - or sometimes ordered chaos - to produce a systems’ value. Thus with decomposition in mind, each concept in the architecture in Figure 2.2 can be defined using standard IEEE terms, IEEE (1990). The individual system hierarchy nodes define as;

A System that is a “*collection of components organised to accomplish a specific function or set of functions.*”

A Subsystem that is a “*secondary or subordinate system within a larger system.*”

A Component that is “*one of the parts that make up a system. A component may be hardware or software and may be subdivided into other components.*”

Further to these definitions, recent interest in specifying system viewpoints appears in systems research, Dori et al. (2019). In his view on system definition, Dori references a decade-old question relating to the arrangement of “elements” that make up a system. The definition of system seen in Dori et al. (2019) is penned as *“an arrangement of parts and elements that together exhibit behaviour and meaning that the individual constituents do not.”*

A precise systems meaning and value of IoT has become broad and sometimes overtly conceptual. However, some essential components support the definition as an interconnection of unique sensors and actuators into a private or public communications network. The body of knowledge for IoT exists as a set of agendas that focus on sensor technologies and the creation of intelligent devices that communicate amongst each other to serve a purpose. The core focus here is to enable a purpose that is greater than the sum of its parts. This view reflects the constructs of a system in the true sense of the definition. A belief in general systems theory is that a multi-levelled systems hierarchy exists. This hierarchy evolves from static structures to simple dynamics, towards self-regulating behaviours, and then finally to a system set that determines its very own end state via the understanding of “absolutes” that exhibit systemic behaviours. This belief is a view cited in Dori (2002), in his views on General Systems Theory. A survey of a set of definitions that places IoT within a system hierarchy state the IoT’s overall purpose, extracted from primary reference sources. These definitions show overlapping areas of concern.

The definition of IoT as seen in a strategic research agenda by Vermesan et al. (2009) is: “an integrated part of future internet and could be defined as a dynamic global network infrastructure with **self-configuring** capabilities based on standard and interoperable **communication** protocols where **physical and virtual “things” have identities, physical attributes, and virtual personalities** and use intelligent interfaces, and are seamlessly integrated into the information network.”

The definition of IoT, as seen in a second strategic research agenda, Ailisto et al. (2011), is: “a global network and service infrastructure of variable density and **connectivity** with **self-configuring** capabilities based on standard and interoperable protocols and formats. IoT consists of **heterogeneous things that have identities, physical and virtual** attributes, and are seamlessly and securely integrated into the Internet.”

The definition of IoT as seen in a research overview by Miorandi et al. (2012), is that: “(IoT)

is broadly used to refer to both: (i) the resulting global network interconnecting **smart objects** by means of extended Internet technologies, (ii) the set of supporting technologies necessary to realize such a vision (including, e.g. RFIDs, sensor/actuators, machine-to-machine communication devices, etc.) and (iii) the **ensemble of applications and services** leveraging such technologies to open new business and market opportunities.”

The definition of IoT as seen in an IEEE ecosystem study, IEEE (2015), that focused on the IoT market, technology and standards is that: “IoT refers to any **systems of interconnected people, physical objects**, and IT platforms, as well as any technology to better build, operate, and manage the physical world via **pervasive data collection, smart networking**, predictive analytics, and deep optimization.” Additionally, the definition includes that “for the most part, IoT devices will be **self-configuring** and adaptive to reduce the need for human interaction.”

To distil the reasoning behind the creation, understanding, and evolution of the IoT-Ce, the system’s main characteristics, as seen in Dori (2019b), should deliver a representation of practical modelling constraints. These characteristics are the *Problem Occurrence*, *Purpose*, *Primary Function*, *Enablers*, and *Environment*. A strongly typed OPM extract filter that adheres to a template-based approach for OPM - as seen in Soderborg et al. (2003) - applies the necessary object, process, and state guidances to extract and bind the system structure. The standard gerund identification out of the definitions by Vermesan et al. (2009), Ailisto et al. (2011), Miorandi et al. (2012) and IEEE (2015) yields process elements as candidates for argument. These are; **Self-configuring**, **Interconnecting**, **Networking**, and **Supporting**. Additionally, the gerund equivalents - as required by ISO/TC 184/SC 5 (E) (2014) - from these definitions can also extract process elements such as **Identifying**, **Interfacing**, **Communicating**, **Sensing**, and **Adapting**. The extraction of elements that could provide form and function to the system gives rise to the ability to argue existence criteria for decisions related to the system’s representation. For example, the need for the existence of an **Identified** device before the **Networking** process occurs. An existence criterion of objects before processes is a view that represents the existence of an object against which a process can execute a transformation, Dori (2002).

The delivery of a diagram, specification, or model that outlines a critical problem statement applies to a vast structural and behavioural variation in the critical IoT domains of focus. Domains such as; data generation & acquisition, data transmission, data analytics, front-end systems, near-end systems, and far-end systems variation produce higher degrees

of complexity as expected in systems with a large number of integrated components and interactions. A high multiplicity of interconnections, functional aspects, and form constructs in the system specification define the system's complexity. This view of complexity is held by Dori Dori (2011) and in complex systems architectures Crawley et al. (2016)(pp.49). Boundary setting is, therefore, a core part of the definition for a system, as seen in ISO/TC 184/SC 5 (E) (2014).

A system naturally splits into controllable and non-controllable parts. The former applies a systemic view, whereas the latter is termed environmental. A strict constraint exists in that systems modellers cannot change the structure or behaviour of environmental systems even though they interact with the core system. Section 6.1.5 of OPM ISO-19450 indicates that *“The system’s environment shall be a collection of things, which are outside of the system but which may interact with the system, possibly changing the system and its environment. The modeller shall distinguish these environmental things, which are not part of the system, from systemic things, which are part of the system. The modeller is not able to architect, design or manipulate the structure and behaviour of environmental things even though those environmental things may influence or be influenced by the system.”*

2.2.1.2 About Structured Systems Design

Design methods such as structured systems design, waterfall, and agile design have attracted substantial interest in systems research over the last decade. The design and development of software for an embedded system follow a similar structured process, as found in general information technology software engineering practice. However, the embedded systems process requires a hardware-to-software mapping to ensure that hardware specifications can meet software requirements to process and execute tasks assigned to an embedded systems processor. Embedded systems design and deployment suffer from different requirement classes compared to conventional information technology systems, Henzinger and Sifakis (2007). General IT deployments focus on a post-event interaction. Cyber-physical systems implement control based on feedback sensed from the physical user environment. In reality, a physical boundary between the sensed and sensing environment exists. However, a model representing the interaction between physical and cyber extends the real world into something that represents the real world. This representation is called abstraction - a well-known system modelling concept.

Due to the physical boundaries between informatical and physical subsystems or components, there is a need to think about problem occurrence as either systemic or environmental.

Namely, those aspects of a system within the control of the design and those environmental concepts outside of design control. A boundary “separates things that are under the control of the system, its user or designer, from things that interact with the system but cannot be controlled by it.”, Dori (2019b). Thus, thinking about boundaries naturally creates areas of control and modularisation, which suits IoT-Ce design. Things interact within and across boundaries to provide some form of value. These interactions are seen in Figure 2.1. Additionally, boundaries can extend to the concept of boundary conditions into co-design architectures where gaps might exist between the cyber and physical domains such as those modelled in, Mordecai and Dori (2017a). The use of co-design between software and hardware can reduce inconsistencies in design, as indicated in Stankovic (2014).

Therefore, a co-design process is needed to structure functional requirements and non-functional requirements, such as dependability. An example of this is evident when designing the performance metrics of a whole product system in Thiele and Wandeler (2005). In further support of this view, co-design for IoT ecosystems is necessary due to the nature of the distributed systems architecture, requiring support for heterogeneous applications and **Front-end System Set** requirements. This co-design view concurs with the views indicated for network control systems with associated diagnosis and fault-tolerant design considerations seen in Aubrun et al. (2010)(pp.10-11). Although current research shows the result of applying specific algorithms to meet a design goal like power efficiencies or increased scalability in Rao et al. (2009), or control system stability in Feiler and Hansson (2007), the body of work on abstracting these findings into a well-structured systems architecture for the IoT-Ce subsystem is still mostly unknown. Additionally, the effect of architecture decisions on IoT application runtime that implement algorithms for different device state awareness models is largely missing. This gap is primarily due to research of device state, which focuses on embedded systems as systemic components in IoT. A primary body of work supporting this view was produced from 2002 to 2017 by the Center for Hybrid and Embedded Software Systems (CHESS) at the University of California Berkeley, Brooks et al. (2018).

Figure 2.3 outlines a design exploration concept that inputs into a shared logical area dealing with mapping, scheduling, and prioritisation. These are all primary concepts in the embedded systems domain. However, the system’s configuration - the arrangement of components - serves as an intermediary process output that enables application resource control to a limited constraint boundary found in applications and associated execution platforms. Furthermore, as the number of modular systems, subsystems, and components fluctuate, the runtime arrangement of these components serves to alleviate overall state failures under

specific failure conditions. This behaviour manifests in total system failure cases, such as those found in Yigitbasi et al. (2010). An iterative design process thus exists for dealing

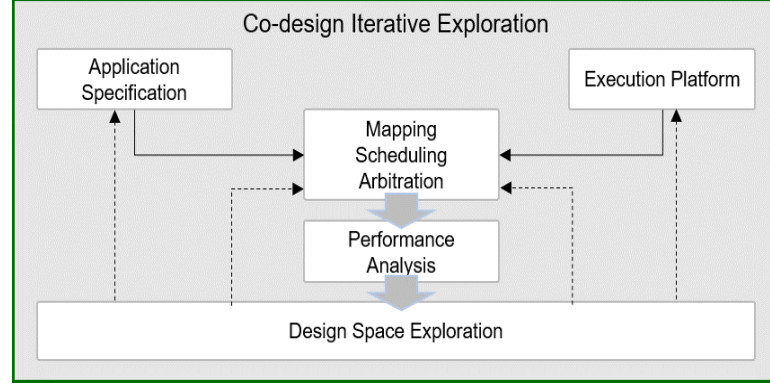


Figure 2.3: Design Space Exploration and Performance Analysis Relationships extracted from Thiele and Wandeler (2005)

with reliability and decisions on the application and execution platform. As seen in iterative design exploration in Figure 2.3, an abstraction of crucial design characteristics exists for embedded systems. The design and deployment of an embedded system follow a defined delivery process. In addition to the process components in Figure 2.3, embedded systems design also considers timing, resource usage, power consumption, and reliability as design requirements, Marwedel (2010).

The proposed approach for delivering systems that meet non-functional requirements in highly complex systems focuses on mapping application requirements to the platforms on which these applications run. Research indicates that software engineering models with specific non-functional and operational requirements handle complexity in embedded systems. However, a reduction of complexity in structured embedded systems is a known factor in embedded systems design. In arguing for hierarchies, abstraction, and co-design complexity reduction in the structured systems design process, Eman et al. (2002), Rumpler and Elmenreich (2006), and Kopetz (2008) apply views on cognitive impacts of understanding systems design. Rumpler and Elmenreich (2006) and Kopetz (2008) states a noted separations of concerns between conceptual components and relationships. Eman et al. (2002) further indicates a decomposable hierarchy to support complexity reduction in embedded systems design. The decomposition of a system into associated subsystems and components is cited in IoT systems architecture (Section 2.2.1.1) as requiring some form of thought on the composition and boundaries in the system. Practical implementation issues for systems that need to conform to boundary-based constraints appear in Laplante and Ovaska (2012).

These appear as;

1. the *selection of hardware and software*, and constraint modelling in a distributed architecture with a focus on concurrency and synchronisation,
2. *high-level programming language nuances* and implications of a compiled machine-language codebase,
3. *application-specific optimisation* utilising robust specification and design,
4. *outlining advantages of open systems* technology and interoperability, and
5. *representation of temporal behaviour* in the specification and design of real-time systems, specifically the measurement and estimation of response times.

A design approach that uses a hardware/software co-design can achieve high interoperation levels between software capability and the limits of the runtime environment possible on the intended hardware. This view is present in Marwedel (2010). Given the design space exploration iterations mentioned in Figure 2.3, Figure 2.4 expands a view on embedded systems software design process as a hardware and software input design model. As represented in Figure 2.4, and supported by Figure 2.3, the embedded systems design process uses inputs from specifications, components, and other embedded systems considerations in order to realise a model output that can be validated. Thus, the proposed approach for meeting non-

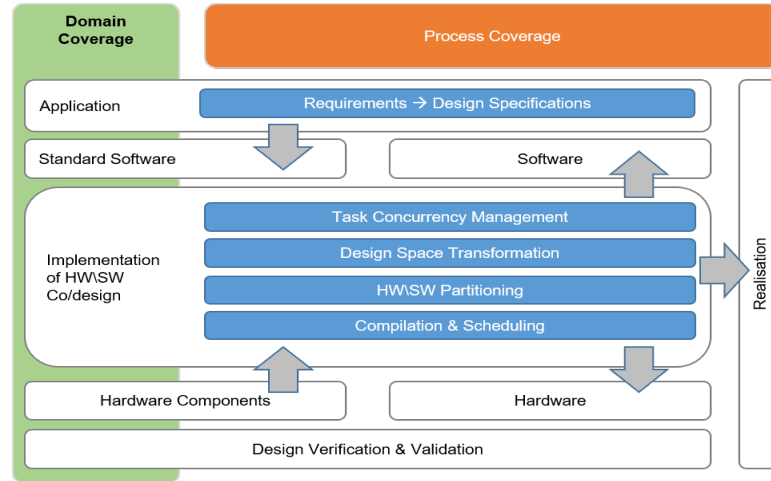


Figure 2.4: An Embedded System's Design Process Components (extracted from Design Flows Section 1.4 in Marwedel (2010))

functional requirements such as reliability structures a mapping of application-to-hardware is a primary transferable argument for IoT-Ce. IoT-Ce is defined herein as containing the

population of intelligent mobile devices containing embedded systems.

In addition to structured design and overall systems product design, Bahill and Madni (2016) outlines the results of a process study to deliver predictable outcomes in a systems design process. SIMILAR, is a set of process activities that align closely with the reasoning behind why and how systems come into delivery. The SIMILAR activities state that a problem needs to be; **S**tated, **I**nvestigated for alternatives, **M**odelled, **I**ntegrated, **L**aunched, **A**ssessed and finally, **R**e-evaluated. These activities iterate in a circular and parallel manner that uses evaluation at every activity within the larger process.

The structured process represented in Figure 2.5 outlines a series of parallel and ordered system design steps. Large-scale complex systems generally rely on the modelling of a system to

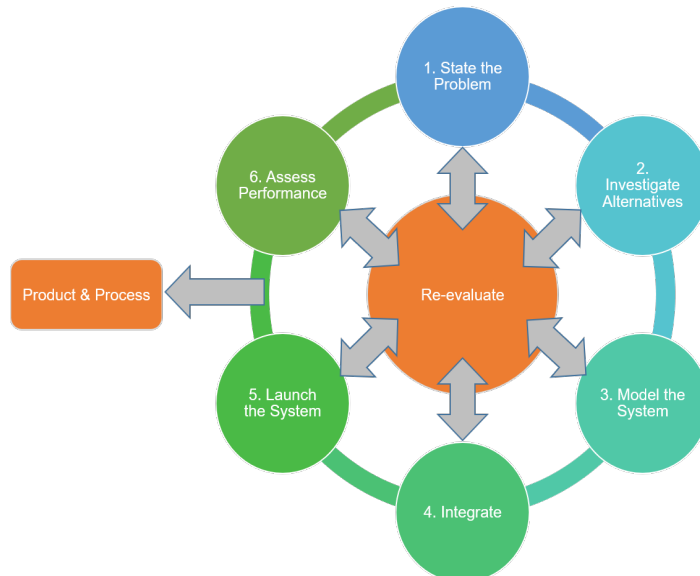


Figure 2.5: The SIMILAR Process extracted from Bahill and Madni (2016)

communicate the intended structure and function of the system. Models in systems design play an essential role in understanding the complexity or, conversely, the relative simplicity of a system. Thus, a preliminary view on the need for a structured systems design process that delivers a set of system models requires;

- *broad systems coverage* due to systems heterogeneity,
- *cleaner boundary divisions* to enable modular systems design,
- *simplification due to variable hardware and software co-design* considerations, and

- a mechanism to *reduce complexity* in various layers in the architecture *while maintaining model integrity* between component hierarchies.

The design requirements for handling a distributed system architecture running IoT applications can manifest in those applications that execute a distance away from the core IoT systems environment- practically on the IoT edge. Validation is needed to effectively identify the decisions that impact variables for single subsystems or multi-node interconnected systems. Given the levels of embedded device demand and its variations, such as those seen in Instructables.com (2017) and Arrow.com (2018), the system domain for the IoT-Ce indicates that IoT device deployments will increase substantially in an eventual standardised form. The proposed standardisation levels in research outline the spectrum of variation in IoT deployments, but more specifically, IoT in the cyber-physical domain, Schoitsch and Niehaus (2017).

2.2.1.3 About Models

The output of the structured systems design process discussed in section 2.2.1.2 produces a system model. Models should accurately represent the environment within which the intended systems will operate. Therefore, a structured systems design process must form part of the IoT-Ce design approach to produce a usable model for validation purposes.

In systems modelling, many asserted properties about the system are, in fact, not properties of the implementation of the system but rather properties of the model of the system. This principle, known as the Kopetz principle, is taught by one of the foremost authorities on CPS design - Prof. Herman Kopetz of TU-Vienna, Ptolemaeus (2014). Properties such as determinism, timeliness, and reliability represent some of these non-functional properties. The creation of concrete models is ordinarily present in some form of modelling notation or language. Several tools and techniques have been introduced throughout the historical period of active systems interest to enable design efficiencies. Tools that cover end-to-end modelling activities for embedded systems and cyber-physical systems implement abstractions using known lightweight implementation languages such as those seen in Ptolemaeus (2014). Others that support core systems engineering constructs have been born out of graphical modelling languages such as UML and SysML. SysML, by the Object Management Group, OMG (2017), is seen as a beneficial modelling approach due to its multi-viewpoint modelling method. Modelling notations such as SysML provides cross-cutting, structural, and behavioural constructs to specify exact unambiguous requirements, OMG (2017). Additionally, architecture analysis and design standards, such as the Architecture Analysis and Description Language (AADL), support the modelling of configurations

at runtime Feiler and Hansson (2007). The syntactic constructs that support a model representation are helpful because they represent structural and behavioural model outcomes, which determines what the model is and what it is not.

Model fidelity and credibility represent the trust in the usability of a model, Cameron et al. (2017). Where fidelity focuses on the degree of accuracy of representation for a real-world system, the model's credibility guides decision confidence in the model. Cameron et al. (2017) forms part of a rich body of work underpinned by Crawley et al. (2016) and the broader OPM ISO 19450 body of work by Prof. Dov Dori. Cameron et al. (2017)(Week 4 Course Notes) presents the following model qualities for good model design;

- **Model Fidelity** - How well the model represents the system's representation in the real world.
- **Model Credibility** - How much trust can be put into the model by decision-makers.
- **Linked to Decision Support** - Support for decisioning in the face of a large decision, or variable, parameter space.
- **Internally Consistent** - Applicable to physical models; however, a set of consistent values should exist, such as Component Transfer Time.
- **Analysable and Traceable** - Easy identification of model and variable dependencies.
- **Availability of Interfaces** - Should have data and output that are accessible via interfaces.
- **Well Formed for Optimization** - Expose necessary information for optimisation activities.
- **Avoid Black Box Optimization** - Optimisation performs poorly if the underpinning model constructs are unknown. Take advantage of the underpinning model structure and features.
- **Data Extrapolation** - Great models clearly outline limitations and constraints to support the model validity and where they are not valid.
- **Validation** - Model must meet intended requirements and expectations. The organisation of the model should be hierarchical in its presentation of information.
- **Verifiable** - Model must be verifiable apart from the outputs that it produces.

Figure 2.6 classifies three classes of attributes representing model coverage, trust, and usability for qualities of good model design.

Models verify and validate the system, subsystem, component, or feature across multi-

Model-based Quality Attributes		
Coverage	Trust	Usability
Complete Relative to Scope & Intended Purpose	Model Fidelity	Availability of Interfaces
Clear Scope	Model Credibility	Well-formed for Optimisation
Understandable & Well-organised	Linked to Decision Support	Black-box Optimisation Avoidance
Elegant & Reusable	Internally Consistent	Support Data Extrapolation
Verification & Validation with Models	Geared for Analysis & Traceability	Validation & Verifiable

Figure 2.6: Model Attributes for Good Model Design adapted from Cameron et al. (2017)

ple use contexts. An extract from the lecture notes in Cameron et al. (2017), indicates the hierarchical nature of the models in system design. Additionally, product feature validation occurs via statistical process control and boundary testing. The importance of boundary conditions in IoT is mentioned in a preceding section on IoT systems architecture (Section 2.2.1.1).

The types of verification and validation models can thus vary from being computational to physically produced prototypes. As indicated in Figure 2.7, the system, subsystem, component, and feature model types can all realise by simulation or some form of assembly. Furthermore, in their view on systems, Høyland and Rausand (2009)(pp.75) defines the system as containing elements, boundaries, interactions, and boundary conditions as the concepts that bind the system into internal and external components. In support of the eventual emergent function of the system, the form needs to contain sufficient information against which a functional component can map. As seen in Høyland and Rausand (2009), a functional block represents an element of the system, whether it is an individual component or subsystem within the system boundary. Finally, on model definition, the IEEE views verification and validation as the processes that provide the basis for determining the correctness, completeness, and compliance of the system to its requirements, IEEE (1990).

1. **Verification** of a model must be verifiable outside of the outputs that the model

System Hierarchy	Model Stage	Model Type
System	Models	Discrete event simulation
	Prototype	Early pilots
	Production	Launch (For Product V&V, Reliability testing can be used)
	Post-production	Key Performance Indicators
Sub-system/Function	Models	Digital mockups and Assemblies
	Prototype	DFA Analysis
	Production	Pilot (For Product V&V, Reliability testing can be used)
	Post-production	-
Part/Component	Models	Process simulation (For Product V&V, FEA can be used)
	Prototype	DFM
	Production	Design of experiments
	Post-production	Key Performance Indicators
Feature	Models	Process simulation
	Prototype	Prototype mfg process
	Production	SOPs/Pilot
	Post-production	SPC/Process Auditing

Figure 2.7: Model Options for Process Validation and Verification Activities. Extract from Week 4 V&V Course Notes Cameron et al. (2017)

produces, Cameron et al. (2017). Incose (2018) further define these as the top-level requirements, which fulfils the customer need, and that approach against which the system realises is correct.

2. **Validation** is the process of determining whether the model meets the intended use of the model Cameron et al. (2017). Incose (2018) further defines this as a tool to determine if the system value has been achieved.

To ensure the correct representation of systems, validation and verification of simulation models are useful in outlining the behaviour of the system under constraints. Sargent (2010) positions several validation techniques to decide model validity. These categorise on both ends of the model spectrum, namely from conceptual models to historical data-based validation models. Furthermore, Sargent (2010) indicates parameter variability using sensitivity analysis that supports qualitative and quantitative analysis. Fragmentation of verification and validation models stretches into CPS modelling with research as recent as 2017 Zheng et al. (2017). Zheng shows that CPS developers indicate trial and error activities that prioritise over verification and validation. Model-based approaches such as OPM ISO-19450 are not considered by Zheng et.al., even though the finding related to “more expressive and

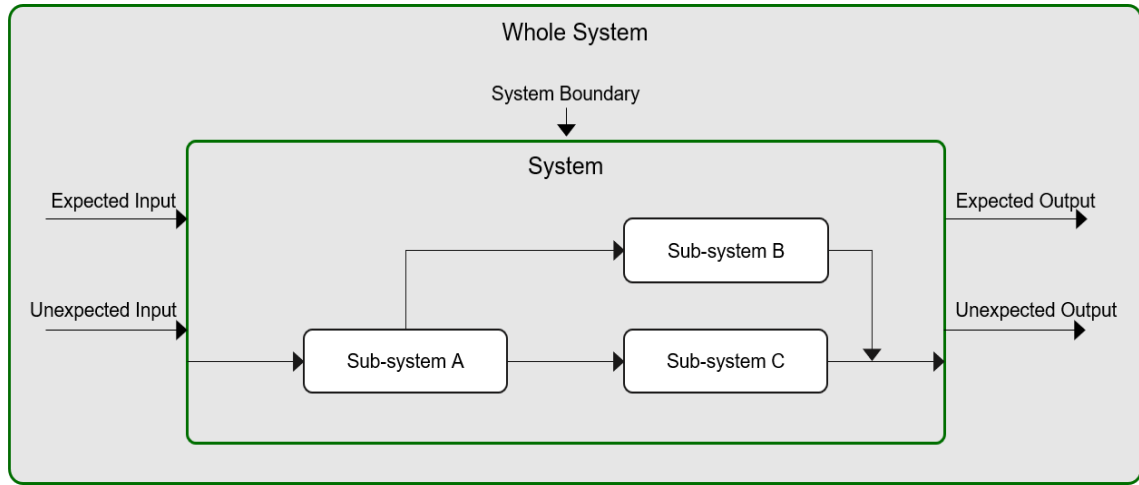


Figure 2.8: A System, Boundary and Interfaces. Extracted from Høyland and Rausand (2009)

integrated models” for CPS is presented by the author. Although the author specifies both verification and validation, validation is not discussed in detail in the state of the art view on CPS modelling.

2.2.1.4 About Model Simplification & Analysis

An assertion exists that it is not the system itself that must be simplified but rather the model representing the system. Essentially, the challenge in complexity is really about simplifying the representation of the system to attain cognitive learning benefits, Kopetz (2008), and Rumpler and Elmenreich (2006). Further simplification of models by reducing the conceptual relational aspects of the system structure is evident, Rumpler and Elmenreich (2006). The view on using system concepts such as boundaries and interfaces reiterates in Rumpler and Elmenreich (2006). Thus, the reduction of complexity as cited in Kopetz (2008) resolves to a notion of a system boundary as a means to manage system failures. This view supports the system input and output structure in Figure 2.8, which forms that basis of classical functional analysis seen in Høyland and Rausand (2009)(pp.78). The reduction of model content to a hierarchy with clear boundaries thus forms a principle for the IoT-Ce systems architecture because hierarchy and boundary are primary systems characteristics. The simplification target thus balances model precision and representation of the real world.

Modelling of the IoT-Ce, whether physical or informatical - can ppear in a single model or multiple closely related models. To reduce ambiguity in model communication, it is preferred to keep the model in a single repository without executing any translation of the model. In Model-based Systems Engineering (MBSE), the Object Management Group’s

SysML standard contains modelling artefacts that deal with standardisation and modelling of solutions to systems engineering problems, OMG (2017). However, a simplification of systems modelling constructs is argued for in Dori (2002), Dori (2011), Dori (2016a), Dori (2019a) and Dori (2019b). The culmination of this work has produced the OPM ISO-19450 standard for automation systems and integration, ISO/TC 184/SC 5 (E) (2014). The standard lays the foundation for ongoing modelling between competing modelling approaches and standards. Al-Fedaghi and Alahmad (2017) cites a comparison of SysML, UML, and OPM done by Mordecai et.al. As indicated in Al-Fedaghi and Alahmad (2017), Mordecai finds that these standards contain both structural and behavioural aspects. OPM is however set apart from SysML and UML, as processes are seen equal model constructs to objects and states, Dori (2011) and Dori (2016a).

In his proposed convergence of UML and OPM, Al-Fedaghi produces a flow machine that contains a strict set of states that transforms inputs and outputs, Al-Fedaghi and Alahmad (2017). In simplified form, the differences propose as alternatives; however, based on the flow machine constructs, it appears to be an instantiation of an OPD. Certain standard-guidances are however neglected in Al-Fedaghi and Alahmad (2017). Such as the use of gerund verb placement and implicit process order execution. The latter uses tagged event flows between enablers and objects in OPM, ISO/TC 184/SC 5 (E) (2014). Notably, Grobshtein and Dori (2008) evaluate SysML and OPM using context and system boundary guidances.

In the Dori (2019a) chapter on *System Architecting*, Figure 2.9 is represented using OP-Cloud. The version in Figure 2.9 is however modelled in OPCAT 4.2. The preference for using OPCAT 4.2 is due to this tool’s proximity to the OPM ISO-19450 standard, Dori et al. (2003). OPCAT 4.2 (Object-Process CASE Tool) is a thick client application that applies standard OPM toolbars and symbols. OPCAT applies a real-time model-to-text transformation based on the OPM dual-channel principle while a systems model is specified. In a personal communication with the inventor of OPM (Prof. Dov Dori, personal communication, 20 May 2019), it showed that simulation capabilities were due in a future release of OPCloud - the Software-as-a-Service version of OPCAT. In 2021, OPCloud (<https://www.opcloud.tech/>) supports animated simulation and model debugging. Additionally, the tooling and documentation available for OPCAT simulation indicate the use of the OPCAT client, Dori (2016b), instead of the cloud solution. Advances in the cloud solution would not necessarily extend the usability of the primary ISO standard but rather apply the standard using a toolmaker’s conformance certification.

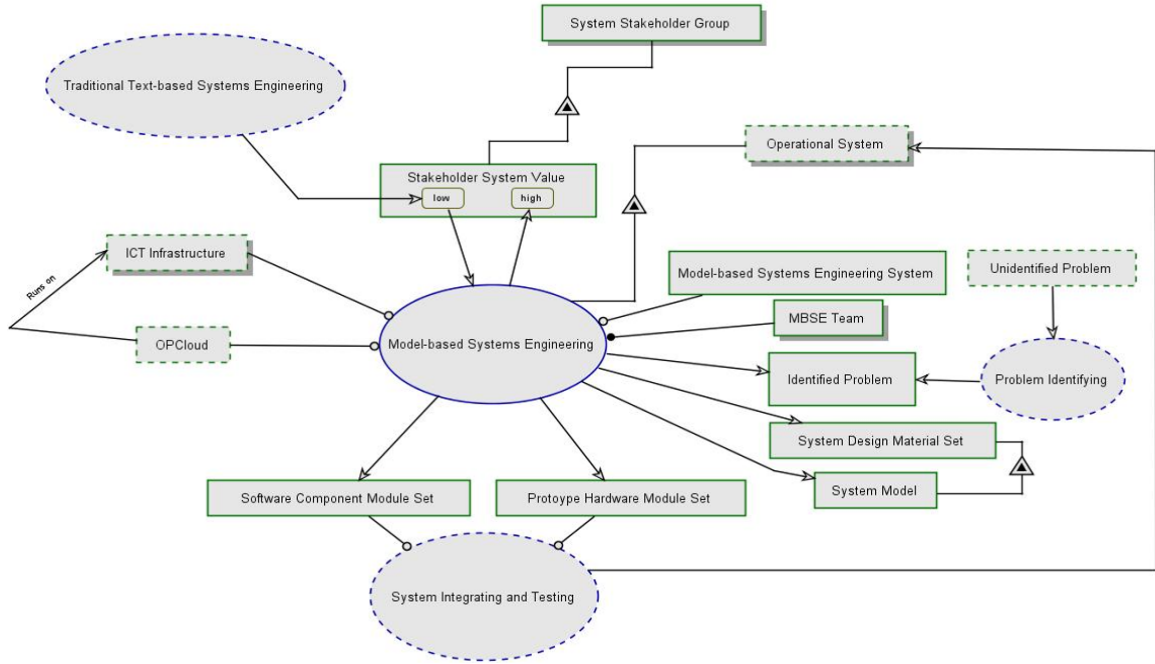


Figure 2.9: A System Model for MBSE adapted from Dori (2019a)

The simplification of the system architecture using a conceptual model (OPM ISO-19450), coupled with IoT-Ce requirements, applies an intended use of the model as an enabler to accurately represent the system architecture and subsequent testing of the system. Essentially, the conceptual model of the system validates using a system test. The evidence in support of the use of OPM ISO 19450 has seen an increase in recent literature. The use of OPM, above conventional model-to-text specification, has found its use in work related to security protocols in Mordecai and Dori (2014), requirements engineering in Mordecai and Dori (2017b), complex system integration in de Matos (2018), and complex system integration with constraints in Li et al. (2019). In stating the systems architecture and subsequent decision model, partial conformance requirements to ISO 19450 ISO/TC 184/SC 5 (E) (2014) is summarised as containing;

1. symbolic conformance to objects, processes, and states,
2. symbolic conformance to OPM syntax and semantics in cases of procedural link usage,
3. symbolic conformance to all other link types such as transformation links, consumption links, enabling links, control links, and condition links,
4. symbolic conformance to structural links and cardinalities and
5. symbolic conformance to logical operators.

In OPM, the modelling of a systems most upper level conceptual model (System Diagram (SD) Level 0) drives five system realisation factors, Dori (2019b). These are;

1. **Purpose**, which describes the systems reason for existence. It consists of a group or set of **objects** exhibiting a main **attribute** containing **states**, and a main **process** that transforms the **attribute** between **states**.
2. **Function**, which describes the main beneficiary value realised from the system. For example in the case of modelling a social system such as a conference, the main process (**Conference Occurring**) transforms **Business Cooperation** from **loose** to **tight**, Dori (2019b). The function is defined as having a main **process** that acts upon an operand, or main **object**, of the system. One of the key architecture analysis guidances is the use of gerund verbs in the OPM ISO-19450 standard on process naming, (section B.6.3) ISO/TC 184/SC 5 (E) (2014).
3. **Enablers** such as an **agent** or **instrument** that interacts with the system.
4. **Environment** which represents the external environment that impacts the system under control
5. **Problem Occurrence**, which describes the reason behind why the system is needed. In the case of the conference system, a decline in business (**Business Declining**) is the reason why the conference system is needed. Similarly, an increase in battery inefficiency (**device battery efficiency decreasing**) would give rise to a system that should increase battery efficiency.

These characteristics create the baseline for a framework through which various analysis is performed to checkpoint the understanding of the systems architecture. Systems architecture maturity forms through the use of the OPM Template approach indicated in Soderborg et al. (2003), cited in Dori (2011). The template-driven approach to modelling systems considers the **what** (emergent value) and **how** (architecture) of systems. Namely, the structure and behaviour that result in the function of the system under specification. The **what** relates to the desired outcome or value of the system, whereas the **how** reflects the structural and behavioural result of a function. For example, a simplified model that outlines the characteristics of MBSE using OPM appears in recent course work developed by the Israel Institute of Technology, Dori (2019a). The **what** in this case is the low value of **stakeholder system value** due to **traditional text-based systems engineering**. The **how** reflects the introduction of **Model-based Systems Engineering** which changes **stakeholder system value** from **low** to **high**.

Figure 2.9 indicates that the **Model-based Systems Engineering** process is the main process that changes the primary operand **stakeholder system value** from **low** to **high**. The **problem occurrence** indicates that **traditional text-based systems engineering** yields **low stakeholder system value**. The main outputs of the primary process is a **Software Component Module Set** and a **Prototype Hardware Module Set**. Figure 2.9 presents a model that simplifies a fairly complex process.

2.2.1.5 About State

The modelling of system state is known to underpin the understanding of dynamic behaviours in a system. The IEEE (1990) defines state as “a condition or mode of existence that a system, component or simulation may be in”. Additionally, this definition alternates to referring to the postulation of values at a given point in time by the “variables that define the characteristics of a system, component, or simulation”, IEEE (1990). In proximity to a systems engineering view on state, Incose (2018) sees system state as “a set of attributes at a given time”. Thus, the arrangement of the system at a given time reflects the state at a point in time. According to Incose (2018), a change in the systems environment from one state to the next shows as being;

- Static, containing a single state that exists with no change,
- Dynamic, containing multiple possible, stable states, and
- Homeostatic, contain a static state composed of dynamic elements that adjust using internal state management.

In modelling system state, SysML focuses directly on state machine package definition through suitable usage of symbols in modelling problem state resolution, OMG (2017). The OMG define the state machine as a representation of behaviour as the “state history of an object in terms of its transitions and states”. Modelling notations such as SysML provides cross-cutting, structural, and behavioural constructs that aim to specify detailed requirements, OMG (2017). Structural and behavioural constructs represent the slowly changing structure and purposeful initiation of change of the modelled system, respectively. Some SysML constructs that support the representation of a model are;

1. **Structural Model Elements** for creating different views and viewpoints to support cross-domain usage of the model. As specified in OMG (2017), “a view is a model element representing a real-world artifact that can be presented to stakeholders, whereas a viewpoint is a specification of the conventions and rules for constructing and using a view to address a set of stakeholder concerns”.

2. **Structural Blocks** that represent a single unit of system description,
3. **Structural Ports and Flows** that represent the interconnecting units between defined system boundaries,
4. **Structural Constraints** that enable the integration of analysis for non-functional requirements such as reliability and performance,
5. **Behavioural Activities** that represent the inputs, outputs, sequences, and conditions for coordinating other behaviours,
6. **Behavioural Interactions** that represent the interactions between entities,
7. **Behavioural Use Cases** that describes the usage of a system (subject) by its actors (environment) to achieve a goal that is realised by the subject providing a set of services to selected actors, and
8. **Behavioural State Machines** defines a set of concepts to model discrete behaviour through finite-state transition systems.

Where most of the model archetypes have found their way in model-based design using concepts such as model-to-code transformations, a few of the model types have found more specific usage paths. Simulation and executable code is an essential usability. This usability is primarily due to the range and robust structure of model types that reduce the distance between specification and validation. State machines are a specific example that has a core focus on control in hardware systems, Wagner and Wolstenholme (2004a). Over the years, it has adapted for use in software engineering. However, there is a general misunderstanding of state machines in software design, Wagner and Wolstenholme (2004a). For the most part, model selection uses a top-down approach that starts with requirements and ends in some form of code-level specification. The verification of such a model can shift tremendously due to user requirements based on changing validations and implementation level code fixes. A state machine is one of the oldest known formal model types for sequential behaviour - executing transitions based on historical knowledge to advance system state. A state machine has a sound theoretical basis and is thus less prone to misrepresentation, such as the misrepresentations seen in heuristic models, Wagner and Wolstenholme (2004a). In considering classes of state machines, two variations of the state machine models exist. *Mealy* indicates that state machine outputs are a function of inputs and state, whereas *Moore* indicates these outputs as a function of state only, Wagner and Wolstenholme (2004a) and Laplante and Ovaska (2012). Wagner argues for using a classical state machine implementation where the basis of transition is on boolean response design, Wagner and Wolstenholme (2004a).

Figure 2.10 presents a typical state machine flow. As mentioned in the chapter on *Finite*

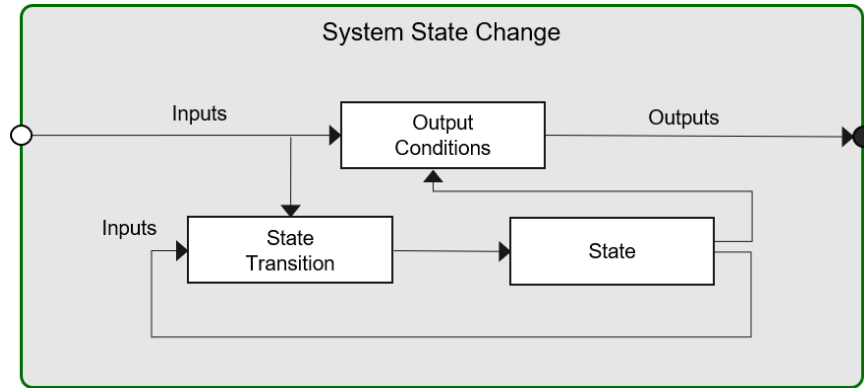


Figure 2.10: A Typical State Machine. Extracted from Wagner and Wolstenholme (2004a)

State Machines in Ptolemaeus (2014), state machines can hold a finite number of states. This model representation provides the necessary constructs to outline the finite states of components in the system, such as component state notification, component state transition, and state boundaries. The primary benefit of modelling machines with finite states is using a set of model constraints that govern multi-technology architecture, such as those possible with abstracted smart IoT-Ce device architectures. Wagner's view on state machines are challenged by Robinson (2004) and then countered by Wagner and Wolstenholme (2004b). The argument and counterargument's primary outcome is the usability of state machines as part of a hierarchy with well-defined layers, interconnections, and accessibility between hierarchical nodes.

The state of a system, subsystem, or component, appears in behaviours seen in everyday interactions. For example, checking the level of drinkable water in a bottle. The bottle exhibits a volume. The water quantity within the bottle can estimate to full, half, nearly half, almost empty, and empty. The level can roughly describe the state of the liquid volume in the bottle. At a conceptual level, there would rarely be a statement that indicates the volume of water remaining is 422ml. That is unless the milk bottle is instrumented to achieve granular measurements. In a system where granularity is a problem, the **problem occurrence** states as *water volume measuring inaccurate*, for which a system would need to yield a value of *increasing accuracy of water volume measurements*. At a point in time, the state of the system is static until a process changes the operand - such as *Water Drinking*, which changes the volume from half to below half. For a system to remain bounded, a finite state machine can provide a fixed number of states (such as those mentioned earlier;

full, half, nearly half, almost empty, and empty) within which a unique set of transitions can occur. State transition is the event that changes the state of the system, Laplante and Ovaska (2012).

The core requirements for adaptive control in configurable systems for failure avoidance indicates that knowledge of state is necessary for self-healing and adaptation, Feiler and Li (1998), Cheng (2008) and Swanson et al. (2014). As indicated earlier, the understanding of state transition is not only for modelling such as OMG (2017), but also in dealing with the integrity of the modelled system. In addition to classical state management seen in Wagner and Wolstenholme (2004a), classical intermittent state awareness is a known aspect of failure recovery in constrained systems, Kopetz (2008). In architectural design considerations, a survey of embedded and CPS design literature outlines the culmination of futuristic CPS design. This design evolves towards an ecosystem where physical devices abstract into the information domain. Essentially, in CPS, these physical components are designed to support a specific service or need. Thus, the final product that delivers value to an end-user is essentially built on physical components underpinning abstracted processes used to present information back to a user. This position on system value aligns with the definition of embedded systems as “information processing systems” enclosed into larger products or systems, Marwedel (2010) and Marwedel (2014). An example of value delivered to a user is the presentation about the state of the system. A cyber-physical approach to systems defines dynamic and static behaviour, more specifically runtime and design time characteristics. The notion of state and state machine transition presents core dynamic behaviour as indicated in Seshia and Lee (2012), and appears as a fundamental model construct in CPS, as seen in Marwedel (2014). State is a result of past system behaviour, Seshia and Lee (2012).

A broader appreciation for autonomous system’s thinking indicates that adaptable systems enforce process cyclical state improvement using standard control features in correcting errors and behavioural state changes in systems, IBM (2005) and Villegas et al. (2017). In the MAPE framework, the definition of the Monitor phase focuses on collecting data from a managed resource such as a component in a larger system composition. Cheng (2008) summarises this as a phase in which extraction of information, such as properties and states, occurs from several data generation entities such as application code or network link probes. The Monitor phase is further defined as having correlation tasks that may aggregate data from several resources and forward this data to the framework’s analysis phase. Therefore, monitoring of a system is an important precursor to the analysis of the reason behind failed system states. These states represent the life of the functional block up to a system failure

point - the survivorship period of a functional block, Høyland and Rausand (2009)(pp.18).

In modelling state in the IoT-Ce systems architecture, the definition of state in the OPM ISO-19450 standard, ISO/TC 184/SC 5 (E) (2014), segments the handling of system state into three classes. These are;

1. “The initial state of an object shall be its state as the system begins operating or its state upon generation by the system during operation”,
2. “The default state of an object shall be the state in which the object is most likely to be upon random inspection”, and
3. “The final state of an object shall be its state as the system completes operation or its state upon consumption by the system during operation”.

OPM defines zero or more initial, default, and final states. Diverse combinations of system state across each class can also exist. In his sections on flow control, Dori (2019a) indicates that the final state, once reached, is immutable even though a final state need not exist.

The systems architecture approach in Chapter 1 indicates;

- using the core OPM conformance requirements to model the ISARC by answering key system delivery questions related to *purpose*, *primary function*, *enablers*, *environment*, and *problem occurrence* as seen in Dori (2019b),
- *maintaining hardware-software model co-design considerations* as indicated in the CPS body of knowledge which supports ED access abstraction as seen in Lohstroh and Lee (2015) and Latronico et al. (2015), and further underpinned by Domer et al. (2009), Marwedel (2010) and Marwedel (2014), and
- *using a structured decision process* to deliver *parameter-based decision outputs* as seen in Bahill and Madni (2016), Kazman et al. (1998) and Washizaki and Tei (2019).

The systems architecture background in Chapter 2 produces further appreciation for the concepts in Chapter 1 as;

- *a systems approach towards IoT*, which indicates a semantic relationships between the definitions in systems architecture that apply to the IoT-Ce.
- *a structured systems design discussion* that exhibits the notion of a design space that serves to position a hardware-software codesign to meet the specification of non-functional system requirements.

- a series of model considerations that present *the characteristics of a suitable model presentation* to fit the IoT-Ce. The background produces a view on model representation within the context of the definitions included in systems thinking, namely a hierarchy for systems, subsystems, and components.
- a model simplification and analysis that presents *a state of the art view on complexity simplification* using OPM ISO-19450. The concise representation of an OPM-aligned model positions five key system realisation concepts; *Problem Occurrence, Purpose, Primary Process, Enablers*, and *Environment*.
- a discussion on state that presents a view on the importance of *understanding state as a driver for system change*. A presentation of classic state concerns is presented with the culmination of state representation using OPM as a state of the art viewpoint. OPM proposes objects, processes, and state as core systems transformation concepts.

2.2.2 IoT Connected-edge

The IoT Connected-edge (IoT-Ce) stands apart from the IoT in that it exhibits systems value away from core cloud infrastructures towards components controlled by end-users. The growth of the number of usable devices over novel system devices sees an elevated utility in intelligent mobile devices when it concerns the IoT-Ce. The edge architecture capabilities discussed in Figure 2.1 refers to the ability of intelligent smartphone devices to play the role of a constrained device (ED) and communication gateway (GW) - termed a **Front-end System Set**. As seen in Yu et al. (2017), conventional IoT system boundaries occur between machine-to-machine, machine-to-cloud, and machine-to-gateway communication models. These models outline a series of system boundaries adapted from the definitions of front-end, near-end, and far-end, Yu et al. (2017). Similarly, the position on a layered architecture that decouples the core cloud-based systems, edge components, and mobile devices is presented in research by Jain and Mohapatra (2018). In the boundaries set by Yu et al. (2017), and Jain and Mohapatra (2018), and additionally by Chang et al. (2019), mobile device components always remain in the edge system boundary, positioned a distance away from the core cloud systems. The front and near-end are essentially closer to end-user controlled resources. In line with capability to support this view, IoT specific frameworks such as the proposed extended EPC global architecture by Uckelmann et al. (2011), creates a broader set of edge capabilities whilst still bound to two core areas of a known IoT architecture, namely the core and the edge.

One could argue that listing a system value such as staying connected in the IoT-Ce raises

the need for lower latent and reliable communication while offloading traffic towards IoT cloud systems. This argument is valid as edge computing requires a strategy to reduce computational stresses and resource congestion handling away from a centralised cloud system, Yu et al. (2017). The offloading of traffic between front-end and near-end components draw a suitable system's boundary. Thus the term connected-edge is placed in the core-vertical thesis argument set. Yu et al. (2017), however, views the placement of traffic away from traditional front-end components characterised as having limited battery life and lossy behaviours, to components that have greater stability in power, such as routers, laptops, and personal computers. This strategy shifts the problem of centralisation from the original cloud centralisation problem to a near-end centralisation problem, which still produces architectural questions related to **Front-end System Set** placement and access. Nevertheless, it exposes the ability to model a single system boundary for ED and GW with a single mobile device. Jain and Mohapatra (2018)(pp.55) sees mobile devices - smartphones - as separated from edge server components.

IoT-Ce components - ED and GW convergence - as seen in generally available smartphone devices - attribute relationships back to IoT definitions. For example, the definition of IoT, as seen by Vermesan et al. (2009), references concepts such as being self-configurable, interoperable, physical, and containing virtual identifiers. Additionally, Ailisto et al. (2011) indicates system heterogeneity, global reach, and enablement of seamless integration to the internet. Miorandi et al. (2012), references the variation of applications and services. Finally, IEEE (2015) indicates the need for data collection and adaptive behaviours that aim to reduce human control. Apart from reducing human interaction, the concepts in the definitions mentioned above relate to generally available smartphones. However, the smartphone as a system value driver has two modes of operation that align with interactivity within general sensing capabilities in mobile devices. A survey focusing on dual-mode mobile phone sensing capabilities appears in Khan et al. (2013). The dual-mode of sensing operation indicates both user-driven and user-less sensing scenarios. These are termed participatory and opportunistic scenarios. Participatory interaction scenarios focus on direct user-controlled interactions, whereas opportunistic scenarios yield additional supporting processing requirements that the user-controlled interaction might not be aware of, Khan et al. (2013).

If an alignment to smartphones (ED and GW convergence) is considered as plausible for the IoT-Ce, it is additionally noted that smart things have fundamental characteristics such as they are uniquely identifiable, they exhibit forms of security, they can communicate freely,

they can sense physical phenomena, and they are involved in information exchange, Vermesan et al. (2009) and Miorandi et al. (2012). Additionally, the body of work by Vermesan and Miorandi, also indicates other domains containing social, autonomous, self-replication, or control capabilities. For IoT-Ce specific capabilities, monitoring parameter state is a vital requirement for those capabilities that increase reliability. Decisions relating to interoperability, ease of use, performance, and flexibility are noted in edge system surveys, Yu et al. (2017). Finally, for IoT-Ce architectures, the movement of intelligence closer to the constrained user terminal is a requirement. This placement indicates an increase in intelligence on a user-controlled smartphone. Vermesan et al. (2009) defines a guideline on the outlook of IoT architecture periods, where adaptivity and cognitive architectures represent greater maturity in IoT, and thus also for the IoT-Ce.

Table 2.1: Internet of Things Architecture Outlook extracted from Vermesan et al. (2009)

Period	Description
Before 2010	IoT Architecture Specifications, Context-sensitive Middleware, Intelligent Reasoning Platforms
2010-2015	IoT Architecture Developments, Network of Networks Architectures, Platform Interoperability
2015-2020	Adaptive and Context-based Architectures, Self-* Properties
Beyond 2020	Cognitive Architectures, Experiential Architectures

Liu et al. (2017) explores the notion of a Mobile Edge Cloud System, with the intent of shifting several processes closer to the IoT edge. Attribute values for low latent use cases via sub-second timing manifests as the requirements for this shift, Liu et al. (2017). Therefore, the complexity of systems can vary from being simple single-node systems operating in a limited boundary environment to multi-node systems stretching local, regional, and even global locations. Figure 2.11 represents the decomposed layered architecture of such a system. A vital note outlined in Figure 2.11 is the separation of concerns between the edge and other subsystems. In particular interest, the edge systems contain both Hardware/Software coupling and components for hardware management. This coupling directly relates to the matter of co-design between software and hardware requirements, as indicated in structured systems design in section 2.2.1.2.

2.2.2.1 Surfing the Mobile Wave

A significant body of work on IoT and CPS implementation has gained momentum over the last few years. Specifically, the use of smartphones to support real-world use cases beyond conventional voice and data communication, text messaging, and social media capabilities that smartphones currently offer. As a core area of research into IoT, there exists a recommendation that IoT capitalise on the concepts in general smartphone use cases. This viewpoint can be seen in viewpoints on architecture and technology dependencies by Stankovic

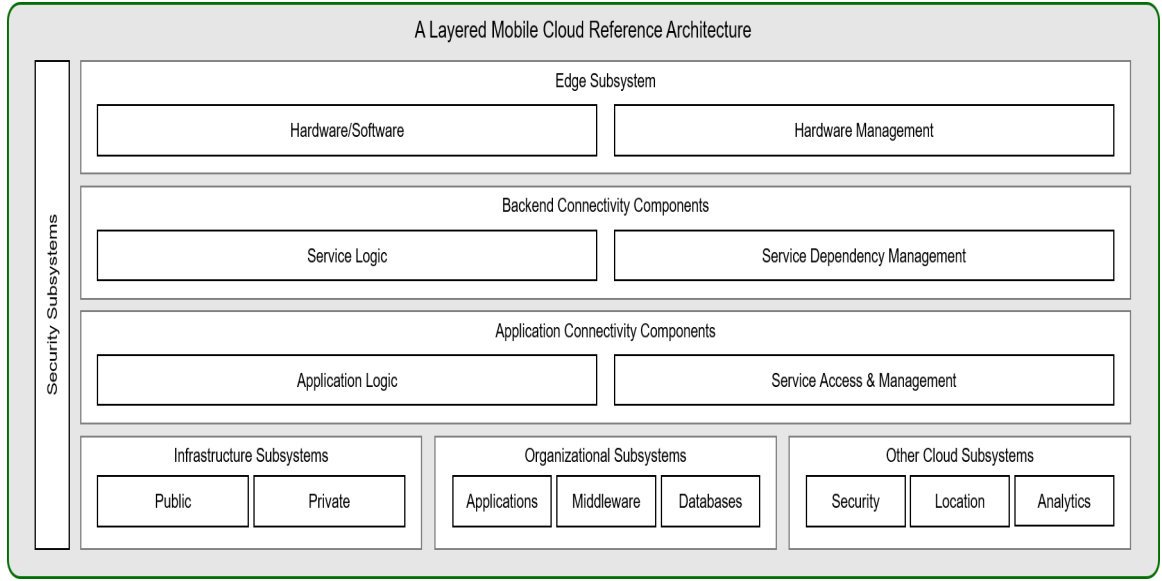


Figure 2.11: A Mobile IoT Cloud Reference Architecture adapted from Sridhar (2013)

(2014). An argument for the use of smartphones are seen in impactful use cases such as human well-being Sinche et al. (2017), behavioural cognition Pei et al. (2013) and patient-centric healthcare applications and services in Horn et al. (2016) and Zhang et al. (2017). Additionally, Khazbak et al. (2017) proposes use cases for optimised data transmission in healthcare. Notably, one of the most insightful use cases shows through the development of tethered mobile application-sensor combinations. EvaTech (2019) prototypes a technology that aims to provide non-invasive thermal measurements for evaluating the possibility of breast cancer in women. The technology tethering approach allows a device housing an array of temperature sensors to attach over Bluetooth to a mobile phone housing an Android application that renders thermal scanning results. As of January 2019, the research was in prototype, but recent searches have shown that Higia Technologies has been re-branded as EvaTech.

Compounding the argument for mobile device usability, strategies by mobile operators provide mass adoption messages that connectivity plays an essential role in daily life. This support positions the true potential reach of IoT-Ce use cases. The support to govern interest for humanitarian efforts is evident in the GSMA's Humanitarian Connectivity Charter, GSMA (2015). This charter sees mobile connectivity as a "lifeline" in times of humanitarian crisis. Mobile networks play an indispensable role in providing access to information and coordination in times of crisis. The opportunity to extract more considerable system value that meets constraints of smartphones housing multiple sensors and constraints of pure smart sensors can be based on the sensor population and combinations of sensor types such

as temperature, accelerometers, light, and humidity. Familiar devices such as the Samsung A and J-series, and more affordable devices such as the Xiaomi Redmi series have a minimum onboard sensor array configurations containing accelerometers, proximity, and gyro sensors, GSMArena (2018). A connected system with multiple sensors offering value to the user holding a mobile phone presents some simple arguments;

1. hypothetically, if it is assumed that approximately 347 million **unique devices** with gateway capabilities exists, IDC (2017),
2. having a ratio of approximately three **sensors** per device, with a single sensor having one or many **attributes** such as the X,Y and Z-axis on an accelerometer,
3. this creates an **ED/GW** population of 347 000 000, and thus
4. the number of **onboard sensors** per **ED/GW** at a ratio of 3 sensors to a single device is therefore 1 041 000 000.

Although these numbers are highly theoretical, the volumes presented can run into astounding upper limits, and therefore a “billions of connected sensor” claim is not a far reach. Thus, forecasts by major technology companies, professional services firms, and academia, Anderson and Rainie (2014) are not entirely incorrect - if you consider a general approach to stipulating IoT growth claims. Additional views for on-person sensors increase the numbers above exponentially; however, the GW would have to be installed on the subject or carried in the vicinity of the in-person EDs. Coupled with the device population shown in Table 2.2, the requirements for treating mobile devices as vital resources drives utility increases for end-users. On the research journey during this thesis, a suitable conceptual population has

Table 2.2: ED Operating System Growth Summary extracted from IDC (2018)

Year	2016	2017	2018	2019	2020	2021	2022
Android	84,6%	85,1%	84,8%	85,2%	85,3%	85,4%	85,5%
iOS	14,7%	14,7%	15,1%	14,8%	14,6%	14,6%	14,5%
Others	0,7%	0,2%	0,1%	0,1%	0,1%	0,1%	0,1%

explored the spectrum of a few externally connected sensors such as those found in Arduino sensor kits and Texas Instrument devices to a larger existing sensor population and network. With a significant number of available devices hosting the Android OS, sensor populations can extrapolate to provide an inclination as to the extent of most control problems. As indicated in IDC (2018), the smartphone population growth on Android surpasses other smartphone operating systems. Table 2.2 outlines an outlook on smartphone OS growth, showing Android as the highest distributed smartphone operating system. A recent trade

war between China and the United States influenced the mobile market, with Chinese corporations considering dropping Android and Google Play support. This change in direction indicates that abstraction away from core operating system functions is necessary for an ever-changing global IoT-Ce market. Recently, Kharpal (2019) indicated that the transition away from Android is happening rapidly.

Mobile devices, or smartphones, or **Front-end System Set** instances as they are known in this thesis, have a specific architecture. For **Front-end System Set** components, a standard

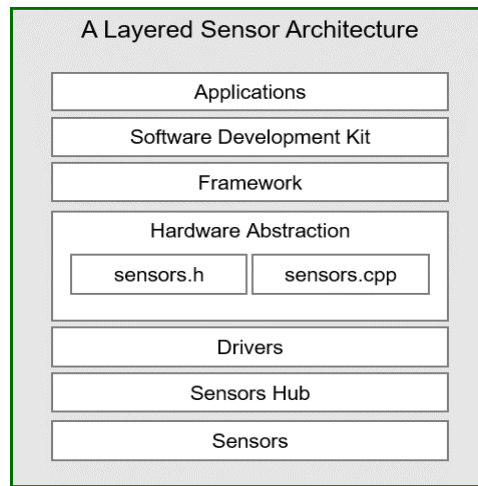


Figure 2.12: The Android Sensor Architecture. Extracted from Android (2017)

Android operating system architecture appears in Figure 2.12. In the Android (2017) architecture, similar architectural layers are present in earlier scope, boundary, and function area diagrams. One could argue that the representation of the operating system's access to the sensor is a credible decomposition from earlier representations in Figure 1.10, that decomposes into Figure 2.1, and then into Figure 2.11, and finally into 2.12. An **Front-end System Set** operating system layers thus manifest as Applications, Software Development Kits (SDK), Framework, the Hardware Abstraction Layer (HAL), Drivers, Sensor Hubs, and the sensors themselves.

- Applications access sensors via the SDK
- The SDK contains functions to list available device sensors and to register to a sensor.
- Framework manages concurrent connections between applications and sensors
- The Hardware Abstraction Layer API is the interface between the hardware drivers and the Android framework. It consists of one HAL interface (`sensors.h`) and one HAL implementation (`sensors.cpp`).

- Sensors and Hubs are the physical chips, and chip controllers, that enable measurements. In many cases, several physical sensors are present on the same chip. For example, some chips include an accelerometer, a gyroscope, and a magnetometer. Such chips are often called 9-axis chips, as each sensor provides data over 3 axes.

In finalising a state-of-the-art position on IoT-Ce components, a view relating to abstracting physical components using high-level programming languages is present in recent research. As noted in section 2.2.1.2, a primary body of work supporting the apex of device access is found within the Center for Hybrid and Embedded Software Systems (CHESS) at the University of California Berkeley, Brooks et al. (2018). Device abstraction is proposed using high level programming languages that implement lightweight protocols to access device states Latronico et al. (2015) and Lohstroh and Lee (2015). Synchronous and asynchronous implementation technologies using NodeJS can run on both client or server subsystems. Libraries such as Express, or full-stack development bundles such as the MEAN (MongoDB, Express, Angular, and NodeJS), enable quicker delivery time due to their lightweight non-compiler nature. NodeJS, as a technology, is a key component in the delivery of deployable units for accessing IoT-Ce components. The specific implementation method appears in the body of research found in the Swarm Research Group at The University of Berkeley, Latronico et al. (2015) and Lohstroh and Lee (2015). Existing NodeJS libraries can be used to display machine interfaces and machine state via the following libraries; `require('os').networkInterfaces()` and `require('os').uptime()` respectively, Linux (2018).

2.2.2.2 Classifying IoT Protocols

IoT protocols aim to provide standardisation for seamless integration into an IoT ecosystem. Protocol selection provides a set of options that compose technical components into a working system. In an attempt to create an ordered classification of protocols shown by Postscapes (2017), the protocol domain contains layers for;

1. **Identification** such as EPC, uCode, IPv6, URIs
2. **Discovery** such as Physical Web, mDNS, DNS-SD
3. **Device Management** such as TR-069, OMA-DM
4. **Semantic** such as JSON-LD, Web Thing Model
5. **Data Protocols** such as MQTT, CoAP, AMQP, Websocket, Node
6. **Infrastructure** such as 6LowPAN, IPv4/IPv6, RPL

7. **Communications and Transport** such as Wifi, Bluetooth, LPWAN, GSM, NB-IoT

8. **Multi-layer Frameworks** such as Alljoyn, IoTivity, Weave, Homekit

The goal of a working IoT system aligns closely with the use of the selected protocol. For example, in connectivity and application integration requirements, networking protocols such as 6LoWPAN, Olsson (2014), and application protocols such as CoAP, Bormann et al. (2014), have been penned to cover the requirements for IoT node connectivity within a local network. Used over standard unlicensed wireless networks, standards like 6LoWPAN offer the ability to support lightweight addressing schemes for nodes connected to a local IP network. This feature enables the assignment of addresses to edge router devices, which supports seamless discovery of device detection within the router's coverage area, Olsson (2014). The seamless enrolment of devices is a key requirement for low-cost, low-power, or lossy devices in an IoT ecosystem. Specific examples of device enrolment is found in Bassi et al. (2013)(pp.239). These enrolment scenarios range in the spectrum of fully automated enrolment - termed automatic joining - to manual enrolment with human intervention. In an earlier view on the IoT-Ce in section 2.2.2, these control scenarios produce user-driven (*participatory*) and user-less (*opportunistic*) scenarios.

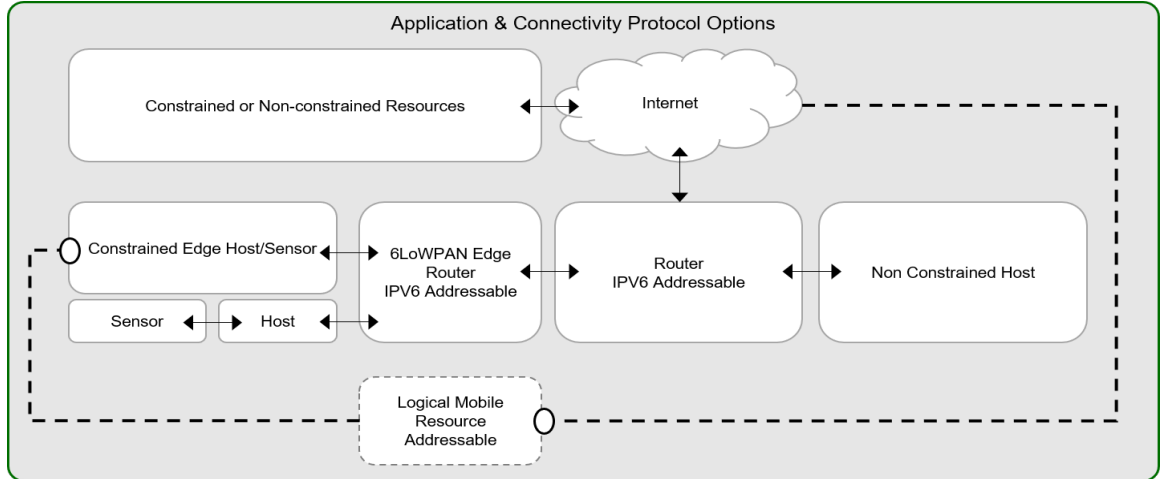


Figure 2.13: Logical Network Deployment for Edge Device Connectivity adapted from Olsson (2014)

Figure 2.13 references the connectivity between internet-facing and IoT device-facing nodes in an IoT network. Devices located in the 6LoWPAN edge network devices' coverage area are reachable and connected in the same sub-network and thus do not require routing but rather an inter-node switching to support data generation and fast actuator control requirements. The sensor and host-sensor (**Front-end System Set**) - entities shown in Figure 2.13 exists in the edge subsystem that needs to be discoverable in the network served under

the IoT network edge routes. A second consideration for the application data transmission loop to be closed, a protocol set also needs to be augmented with specialised application-level protocols such as CoAP. Figure 2.14 presents a further summarised expansion of the architectural components that effectively stack up a new series of protocols and transport requirements for moving data between IoT nodes and moreover to applications that might exist outside of the IoT network.

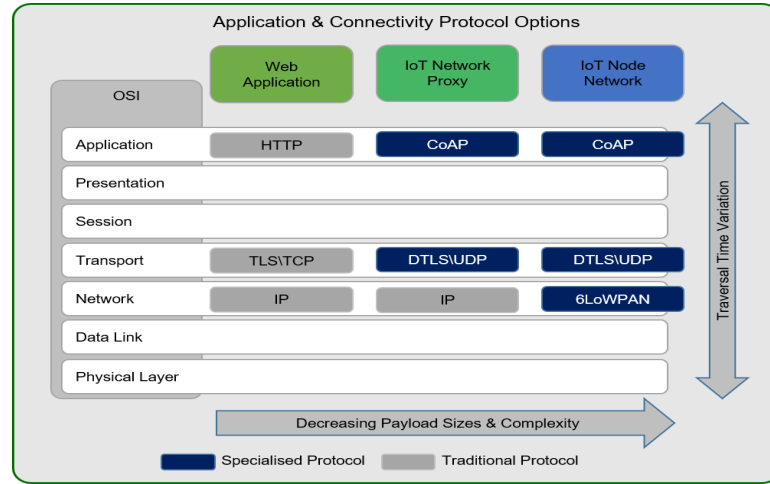


Figure 2.14: A Web Application to IoT Stack Transition adapted from Kostic (2017)

The traversal of communication through these protocol layers has shown to be carefully considered to meet the intended technology goals of IoT, such as the handling of reduced traffic profiles, battery optimisation on edge devices, and increased connection fidelity. Figure 2.14 refers to the strategic placement of IoT protocols with the intended benefit of reducing payload sizes and complexity, Kostic (2017). Protocol selection options can thus provide systemic versus environmental tradeoffs. Every hop, protocol transformation, or data enrichment has an absolute time event cost due to transmission interfaces, transport medium, or application processing requirements.

Under the payload size and latency considerations the following non-functional decision parameters can exist for the protocols represented in Table 2.3. The performance - and

Table 2.3: Protocol Non-Functional Decision Summary. Optional or Required?

Layer	Web	IoT GW	IoT ED
Application	Opt.	Req.	Req.
Presentation	-	-	-
Session	-	-	-
Transport	Opt.	Req.	Req.
Network	Opt.	Req.	Req.
Data Link	-	-	-
Physical	Opt.	Req.	Req.

consequently the reliability - of communication should apply to a decision matrix as the end-to-end communication time might be impacted due to the modularity and communication path for connected edge devices in a mobile proxy connection request. Essentially, given the functional impact of component connections in Figure 2.1 - The aggregated architecture options overview adapted from Eclipse (2016). For example, if the purpose is to select a protocol to serve fewer synchronous interactions, namely UDP over TCP transactions, this can be modelled as a single attribute. Protocol selection decision parameters can thus ex-

Table 2.4: Capability-Protocol Decision Summary

Edge Capability	Appl.	Pres.	Sess.	Transport	Net.	Dat.	Phys.
ED Communication	-	-	-	UDP	-	-	-
ED Hardware Abstraction	-	-	-	-	-	-	-
ED Operating System	-	-	-	UDP	-	-	-
ED Remote Management	-	-	-	-	-	-	-
GW Data Management & Messaging	-	-	-	-	-	-	-
GW Communication	-	-	-	UDP	-	-	-
GW Application Runtime	-	-	-	-	-	-	-
GW Operating System	-	-	-	TCP/UDP	-	-	-
GW Remote Management	-	-	-	-	-	-	-

pose questions about the **Front-end System Set** population. For example, if the selected **Front-end System Set** supports UDP towards an addressable 6LowPAN IoT gateway, does this change any architectural views on reliability? For the architecture decision to be sound, one would consider that UDP is a lightweight transport protocol that sends a message and does not wait for a response from the target systems (GW). Suppose the 6LowPAN IoT gateway in Figure 2.13 is available to receive transactions - that suits perfectly. However, if the gateway is unavailable, would the **Front-end System Set** have to be changed to maintain state, or is system value jeopardised if missing measurements are encountered. In critical control, a missed update could result in less than optimal behaviour within the system. A possible design strategy for control might require both proportional and integral control requirements for a system. Perhaps a feed-forward loop with the resultant error compensation being handled by a component with a larger mean time to failure. Alternatively, doing nothing to increase reliability is an optional strategy as well. The placement of architectural decision options within a large architectural decision space borne out of the IoT body of work cited in Chapter 1, can become large and unmanageable if concepts such as hierarchy and boundary are left unattended. Fortunately, approaches such as viewing IoT-Ce systems architecture within the confines of general system theory are credible.

In defining the IoT protocol selection, Discovery protocols such as mDNS and DNS-SD indicate a “zero-configuration” requirement, Postscapes (2017). The notion of “zero-configuration”

should use a better-suited term such as “zero-reconfiguration”. As argued earlier in the definition of systems, subsystems, and components, a configuration refers to the arrangement of variables representing the system at a point in time. Further, the system’s state is defined as the assumed value of the variables that represent the arrangement of the system. If considered in pure system terms, the IoT mDNS and DNS-SD case for “zero-configuration” would translate directly to “no-configuration” or “configuration-less”. Both terms are not credible because something has to exist before it is changed. This analysis quotes the notion of an object before process and process to change the state of an object. Existence criteria of objects before processes is a view that represents the existence of an object against which a process can execute a transformation, Dori (2002)(pp.61).

The primary approach viewpoint on the IoT-Ce in Chapter 1 outlines;

- the ability to view *a generalisable mobile population* that exposes abstracted interfaces,
- the agreement that *edge intelligence will increase* in generally available smart mobile devices, and
- *increases in the autonomous control* of smart mobile device applications.

The IoT-Ce background in Chapter 2 produces further appreciation for the concepts in Chapter 1 as;

- a viewpoint on *the divergence of core and edge architectures*. Although current frameworks provide holistic IoT frameworks, the primary discussion point in these papers are edge architectures - due to possibilities of increased efficiencies and innovation,
- a direct position of *the large smartphone population as suitable IoT-Ce devices*, as they have the required characteristics to play both intelligent device and gateway roles,
- a direct position that **Front-end System Set** instances *increase utility for general use IoT-Ce* purposes,
- offering *resource access possibilities* for onboard sensor and device functions, and
- a view that *composite protocol requirements require consideration* as either systemic or environmental when approaching systems architecture for IoT-Ce.

2.2.3 Runtime Configuration

“Part of the definition of a subsystem is a list of all the variables that we need to measure in order to determine everything we want to know about the system at that moment in time” Lee Smolin - Time Reborn.

Up to this point, an exploration into a design space for the IoT-Ce shows that both hardware and software considerations are essential in delivering systems that meet some form of system value. Whether the approach to delivering the system, the population that applies the required utility, or the combination of decisions that compose the system, the considerations that drive system specification require structure and behaviour to attain system value. This system value could be; increased reliability, seamless connectedness, adaptivity, or a myriad of individual concepts seen in the recent analysis of the IoT body of work for this study. However, the notion of an establishing reason for these system values should be underpinned by a form (structure) so that a function (transformation) can be mapped to produce emergent system behaviour. The IoT-Ce system exists as an arrangement of variables and basic composable units that produce a flat or hierarchical structure. In Time Reborn, Smolin (2014) notes that a list of variables creates an abstraction called the “configuration of the system”.

To represent the set of *all* possible configurations, an abstract space called the configuration space exists. Each point of the configuration space represents one possible configuration of the system at a point in time, Smolin (2014). The junction of the definition for continuous-time and discrete, measurable time proposes through the theoretical and concrete implementation of the configuration concepts outlined by Smolin (2014), where Smolin outlines the recording of event-based records. This representation aligns with the design of time-based processes in a computer system that can follow either *real* time or *event* time behaviours, as seen in Janert (2013). Thus, the configuration of the system at a point in time represents the arrangement of variables that make up the system and its value. This definition concurs with the earlier definition of configuration assumed via the definition using a software engineering lens, indicated in IEEE (1990). The runtime configuration of the system proposes the arrangement of components during the systems execution phase. This definition also aligns with the earlier views on element arrangement that produces a value. As stated in Dori et al. (2019), a system exhibits through the arrangement of parts and elements that exhibit behaviour and meaning that the individual parts do not - while applying the system realisation tenets of *Purpose*, *Primary Process*, *Enablers*, *Environment* and *Problem Occurrence*. The principle of holism applies that system value is attained by

the whole and not necessarily any individual parts, Skyttner (1996)(pp.61).

In automated configuration problem solving, a configuration is an ordered arrangement of a set of parts, Petrie (2012). With a system defined as a collection of parts that interoperate to produce a function, a system configuration thus defines as the arrangement of attributes of a system to specify the system's behavior supporting a change of its state. This configuration can occur within an electronic computer system when the system is under specification at design time. Alternatively, the configuration can also represent the system at runtime. Runtime configurations exist directly on the equipment running technical processes such as high-end computing processes or even lower-cost mobile applications such as applications running on smartphones. Changes to these configurations are supported on devices via low-level configuration methods through serial interfaces. The serial interface creates an Input-Output (I/O) capability by housing a serial clock pin to receive and control the system's state. Onboard serial registers are supplied with a clock to assist in the measurement and control between process states. Although in-system pin-based configuration standards such as these interfaces are proposed in IEEE (2000), focus occurs in application-level configuration possibilities such as those found in Jacobson (2004). Configuration approaches that abstract functions into higher-level programming capabilities are a user-driven configuration of already configured systems. This view is further supported by the evidence outlined in the earlier section on smartphone device OS abstractions in an increasing mobile device population in section 2.2.2.1.

A key advantage attained in understanding configuration architectures is its inherent flexibility and choice. However, flexibility should offset against reliability as runtime changes to systems can render instability in the system. This type of change over a period can cause the system to settle in a position other than its intended position. The realisation of system purpose, primary process shifts, enablers, environment, and problem occurrence could change if the unintended settling positions are left unchecked. Rychly (2014) views this continuous change as "architectural drift". Shifts that impact the function of a system can be both human-made or automated, depending on the architectural approach used to develop the system and associated components within the system. The system's ability to accept inbound configuration requests given a series of constraints in that system characterises the system's variability, whether pin-based or software-based. Figure 2.15 shows this comparison. A representation of system constraints can utilise a design-time comparison on hardware versus software characteristics, Dart (1991b). Figure 2.15 suggests that a runtime design comparison occurs by looking at the allowable configuration and the capabilities to

Hardware\Software Co-design Parameter Binding		
	Computer-Aided Design	Software Design
Method	Known objects → pin, logic	Unknown objects → files, procedures
Flexibility	Reuse objects → chip designs	Change, build objects → edit files
	Tool-defined process	User-defined process
Coverage	Many views and layouts	Many instances compiled
	Domain-specific	Operating System
Standard	Greater standardisation	Fewer standards

Figure 2.15: Configuration Parameter Variability Characteristics. Adapted from Dart (1991b)

configure. The most notable in the comparison is the method of configuration and flexibility thereof. Within the core ISC-type configuration, several configuration types can change the configured system’s variability. As seen in Jacobson (2004), configuration types can be simple or complex, with numerous variations in this spectrum. Configuration types in Jacobson (2004)(pp.180-181) appear as;

1. **Simple configuration:** “Technicians populated a printed circuit board with programmable devices. Part of the board test procedure includes programming the PLD’s configuration memory with logic patterns. The system configuration is never again changed.” This configuration type has a pre-existence requirement, namely a physical thing on which a configuration can exist.
2. **Field reconfiguration:** “...not time-dependent, but is the most pervasive application. It can provide a technique for updates, bug fixes, and adding new features to digital hardware.” This configuration type has a pre-existence requirement as the system could contain additional constraints such as **Device Installed** and informatical constraints such as **Reference Data**. This configuration offers the flexibility to change some components from being systemic to environmental.
3. **Periodic reconfiguration:** “...is for those applications where there is a regular change of supporting data, such as environmental recording systems or global positioning systems.” This configuration type has a pre-existence requirement, and the system could contain additional informatical constraints such as **Reference Data**.

4. **Frequent reconfiguration:** “...speeds up data processing for applications such as image processing.” This configuration type has a pre-existence requirement for either the object against which the reconfiguration will be processed, namely an image, or the function (object and process) that outputs a processed image. Additionally, the pre-existence of an informatical object such as algorithm which is optimised to generate efficiency or fidelity of processed images.
5. **Runtime reconfiguration containing partial and on-the-fly types:** “applications rely on changing working environment conditions. As these conditions change, so must the application function. Examples of this are; detecting network protocols in a mobile application or interrupting a task with a new service triggered by a security or safety sensor. Some variations within this spectrum include **partial reconfiguration** and **on-the-fly reconfiguration**. These two variations can exist within the spectrum as a subclass of any of the described categories. They may also exist as a separate application class within the spectrum.”
6. **Partial reconfiguration:** “Partial reconfiguration allows reconfiguration of only a portion of the programmable device. This type of configuration allows the development of systems that can remain mostly operational during reconfiguration. Reconfiguration disables only the portion reconfigured. When used correctly, partial reconfiguration allows phased-in reconfiguration of features and tasks with limited impact on system utility”. This type of configuration allows for both physical and informatical changes. The important concept in this type of (re)configuration is establishing boundaries that reduce the impact of change and increase stability in the system.
7. **On-the-fly reconfiguration:** “On-the-fly reconfiguration is typically available only in nonvolatile devices. It allows the programming of the static configuration memory of a device with an alternate utility. Activation of the alternate utility does not immediately occur. Activation occurs when a designer-specified trigger condition occurs. This behaviour allows the development of systems that can preconfigure while running without disturbing the runtime behaviour. When enabled, the alternate system utility activates with almost no downtime. The total downtime is equal to the time it takes to load the static configuration memory into the active memory (typically 100 microseconds)”. This type of configuration shows that design to run time configuration can be constrained by considering the impact of static configuration on the runtime of a system.

The definition of configuration and reconfiguration is a concept that requires clarity due to possible misunderstanding about the functional components for these respective concepts. Additionally, the split between design time and runtime concepts creates a suitably bounded set of functional components. Only by identifying a functional boundary is the architecture able to state what is unique to a view on runtime configuration shared between configuration and reconfiguration. For clarity regarding an earlier argument, and as mentioned in the previous section on IoT protocols, Discovery protocols such as mDNS and DNS-SD indicate a “zero-configuration” requirement, Postscapes (2017). The notion of “zero-configuration” should use a better-suited term such as “zero-reconfiguration”. As argued earlier in the definition of systems, subsystems, and components, a configuration refers to the arrangement of variables representing the system at a point in time. Furthermore, the system’s state is the assumed value of the variables representing the arrangement of the system. If considered in pure system terms, the IoT mDNS and DNS-SD case for “zero-configuration” would translate directly to “no-configuration” or “configuration-less”. Both terms are not credible because something has to exist before it is changed. This analysis quotes the notion of an object before process and process as a means to change the state of an object. Existence criteria of objects before processes is a view that represents the existence of an object against which a process can execute a transformation, Dori (2002)(pp.61).

Compared to the physical nature of the control process found in standards for low-level integrated circuits such as IEEE (2000), larger-scale systems such as mobile telecommunication systems also implement the notion of configuration control in end-to-end systems, Burgess and Mende (2005). Although more abstracted compared to pin-based configuration, the configuration architecture represented in Figure 2.16 logically splits the management, control, and execution layers of a configuration system. This split is essential as it creates a bounded system viewpoint that enables a fundamental step in creating a well-bounded set of functional components in the systems architecture. The requirements outlined in the configuration specifications for a broader use case, such as GSM network configuration, contain several process steps to put a network or resource into operation. As per a network GSM configuration standard, TC-SMG (1996), the process of establishing a network has three main phases; *Installation of the Network*, *Operating the Network*, and *Optimising the Network*. The operational context for configuration management is that the state of control actions required to operate the network must be visible to an operator. In the case of enrolment and management of IoT-Ce *Front-end System Set*, this operator could be a fully automated system actor - noting the difference between participatory and opportunistic system scenarios. The requirements for management enable the execution of simple and timely

control actions, decoupled impact through the understanding of secondary change effects, and detached data consistency strategies, TC-SMG (1996)(pp.14). Finally, the managed resource (MR) state has the following statuses under its resource administration functions - an abstracted finite state machine no less.

- Alarm,
- Procedural,
- Availability,
- Control,
- Standby, and
- Unknown.

Within the managed network, connected subsystems may have functional dependencies, so executing a decoupled resource strategy is necessary. In order to avoid any faults in a system, the configuration will outline resource dependencies, resource impacts, and state propagation, TC-SMG (1996).

The splitting of configuration into logical units is a view that is supported more closely to

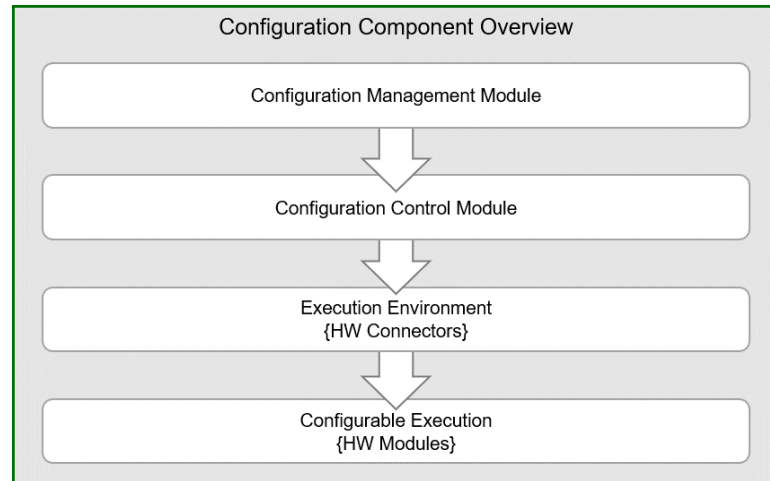


Figure 2.16: Configuration Architecture in an End-2-End Software Radio System adapted from Burgess and Mende (2005)

the IoT-Ce, in the form of configurable wireless sensor networks (WSNs). The configuration of a configurable sensor network appears to split devices (IoT nodes) into two capability sets; Subramanian and Katz (2000) presents this classification as devices used for *discovery*

and another for *dissemination*. This view concurs with the views on gateway and sensor functional boundaries such as those seen in edge computing research, Yu et al. (2017). Self-configurable systems seen in Subramanian and Katz (2000) outlines several assumptions that align with the nature of IoT-Ce; namely, they are heterogeneous, functionally well-bound, and aim to meet power and form constraints. Thus, with suitable component functions and composable subsystem options, the IoT-Ce design and configuration space increases to suitably comprehensive architectural decision space.

The configuration architecture in Figure 2.16 proposes a standard pattern for delivering hardware-controllable software components by subscribing to a series of abstraction principles. Figure 2.17 further outlines an overlap in the configuration architecture by overlaying the hardware and software co-design requirement with the layers represented in Figures 2.4, 2.15 and 2.16. Additionally, the introduction of an abstraction layer between application software and hardware appears in Domer et al. (2009). The inclusion of a middleware component creates the opportunity to externalise system control if decisions relating to internal versus external configuration are required. The integration of configuration management systems provides the necessary decisioning capability to ensure that suitable decoupling levels exist if a configuration management system becomes systemic (in control of the main process) in the architecture. Integration demonstrates as a problem occurrence in configuration systems, Dart (1991a).

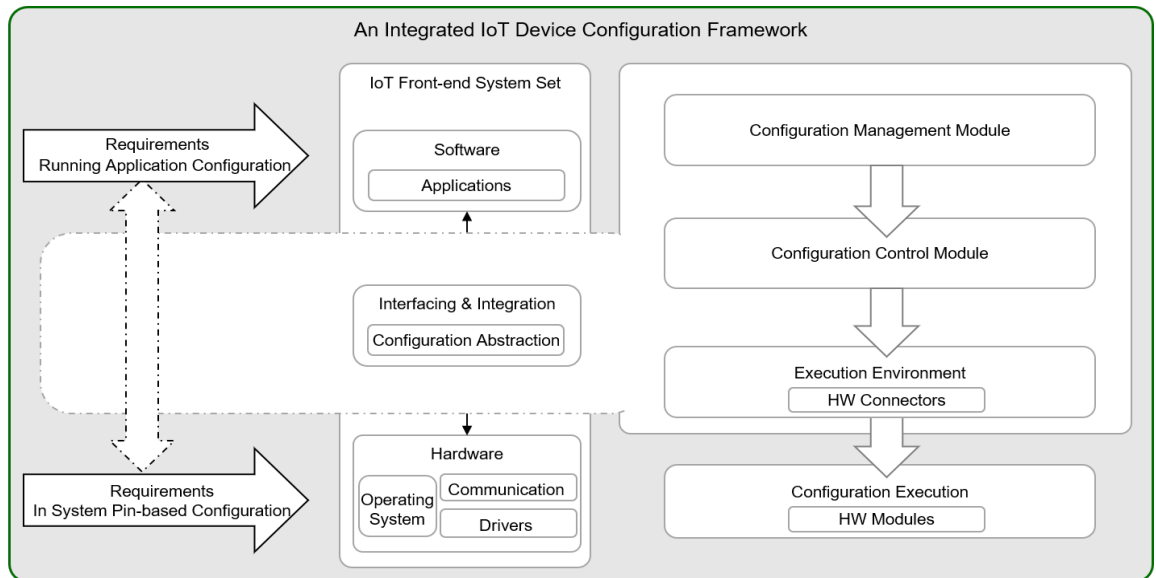


Figure 2.17: A Simplified Layered Architecture for System Configuration derived from Jacobson (2004), Burgess and Mende (2005), Domer et al. (2009) and Marwedel (2010)

Although the current overlay between frameworks such as these can contain different layers, components, and subsystems, it must be noted that the development of decentralised configuration approaches exists since the inception of distributed heterogeneous systems in the early 90s, Burgess (1993). However, continued interest occurs in modular architectures for runtime behavioural change of sensor-rich devices, Fazio et al. (2013), and IT systems integration using IT applications such as Puppet, Chef, Ansible, and other continuous integration technologies. The creation of modular boundaries for configuration in sensor networks is split into the ability of the nodes in the network to be classed into a set of defined applications. Where configuration variation exists in component arrangement, Subramanian and Katz (2000) indicates network topology and layout as the main configuration item against which attributes such as routing and sensor characterises.

2.2.3.1 Component Arrangement

A general view on system reliability for different configurations represents the “connectedness” of components in a system. As seen in Leon-Garcia (2008)(pp.14 and 83), component configurations exist in series, in parallel, or a combination of the two. Figure 2.18 expands the simplified view of the primary system definitions found in the earlier Figure 2.2, where a linear system hierarchy decomposes systems, to subsystems, to components. The decom-

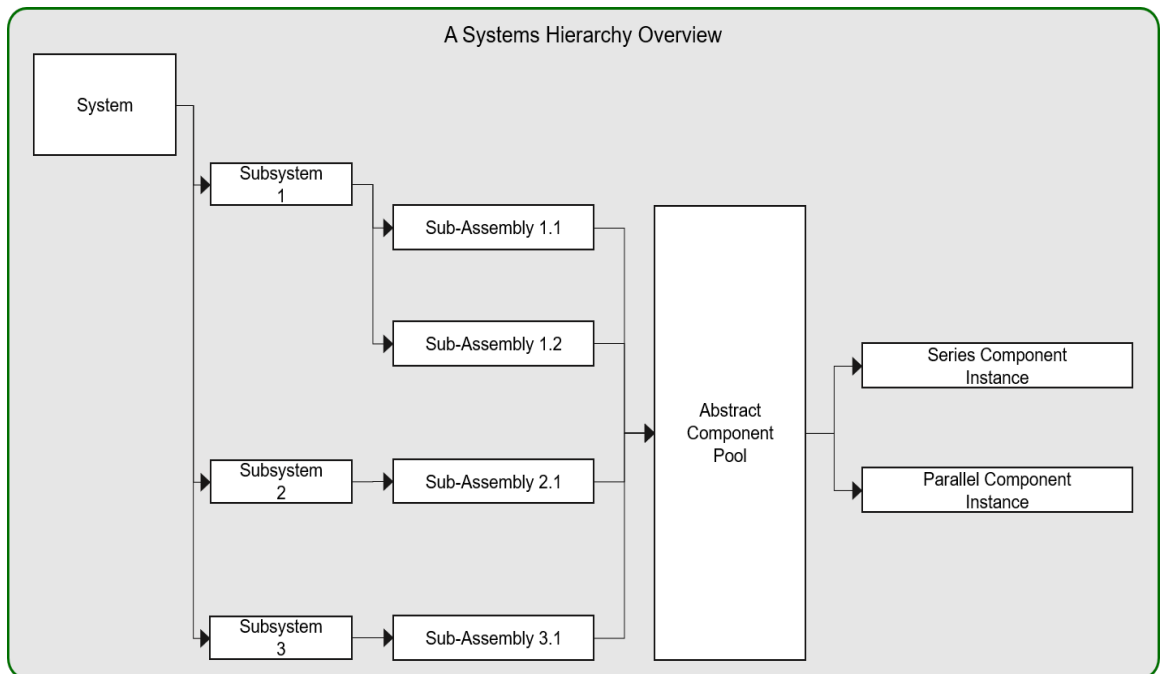


Figure 2.18: Systems Hierarchy Overview

position of system entities into their requisite boundaries enables the compartmentalisation

of these entities to supports earlier arguments for boundary protection and optimisation strategies. Furthermore, in the qualitative analysis of systems, a functional block represents a component or a subsystem within the larger system boundary, Høyland and Rausand (2009)(pp.73).

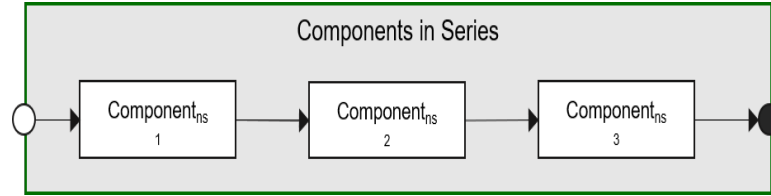


Figure 2.19: Components in Series. Extracted from Leon-Garcia (2008)

For components in series, Figure 2.19 represents several components running in series with a singular input flow. A series-based configuration lends itself to increases in sub-assembly forms that can provide greater reliability for synchronous communication closer to system input channels. However, the probability of end-to-end system integrity decreases as failures in earlier components can compromise dependant flows lower in the processing chain. In OPM, a synchronous process can contain multiple sub-processes that occur in sequence. The processes execute in an implicit top-down order, Dori (2019b). Asynchronous processes occur independently with no specific order. Order can, however, occur using implicit invocations.

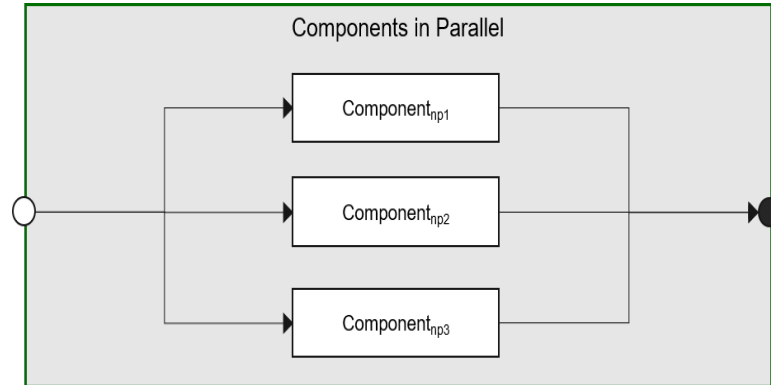


Figure 2.20: Components in Parallel. Extracted from Leon-Garcia (2008)

Figure 2.20 represents several components running in parallel with a shared input. A parallel-component configuration provides the required existence of components to process inputs and produces outputs with necessary state management between parallel components. Similarly, this component configuration provides subsystem and subassembly compartmentalisation capabilities. The essential difference between Figure 2.19 and Figure 2.20 is where the system boundary occurs for components that communicate across boundaries to fulfil

an integrated systems function.

The probability of a component that is available to serve its intended purpose at a point in time is a measure of the reliability of a system, Leon-Garcia (2008) and Høyland and Rausand (2009). This measure is a generally accepted definition for reliability. Furthermore, the limiting aspects defined by quantitative methods in reliability create a defined set of states within which reliability prediction can occur. To limit cascading errors and corrections for those errors, a boundary creates a final state variable up to which the probability of failure can be measured. Coupled with the system, subsystem, and component forms, it provides a suitably limited set of states to measure reliability in system architecture. Four measures of the reliability of faulted components are; the Reliability Function, the Failure Rate Function, the Mean Time to Failure (MTTF), and the Mean Residual Life (MRL).

Component arrangement primarily focuses on individual components with defined functional inputs and outputs. In systems terms, these components hold the object and process transformation needed to deliver a system function. This transformation need not be the primary process that delivers the system value. It could be part of a cluster of functions that, when executed together, provides the actual primary system function. The availability of a vast number of **Front-end System Set** instances with different technical specifications makes the component design process challenging to pin down due to component control enabling the lowest level of system reliability. The arrangement of components can occur either in series or parallel at both design time and runtime. Figure 2.21 represents a runtime flow of two subsystems required to run in series and requires integration in a whole product system. In a view presented by Reichert and Weber (2012)(pp.98), this component arrangement represents a process flow termed a configurable activity placed against a configuration. Furthermore, the configuration that will execute at runtime is determined at design time with optionality about process handling at runtime. Thus, a design-time configuration supports runtime execution with knowledge of the latter, Reichert and Weber (2012)(pp.98). However, the creation of configurable logic blocks ensures appropriate error handling cases, Reichert and Weber (2012). As the systems architecture contains both form and function, the intended system benefit must state somewhat differently from pure procedural process modelling.

Figure 2.21 presents two subsystems that deliver a whole product system. Thus, the allowable and reachable system configuration should be known before runtime execution. This pre-existence condition is similar to the notion of configuration before reconfiguration, as

cited earlier in Jacobson (2004).

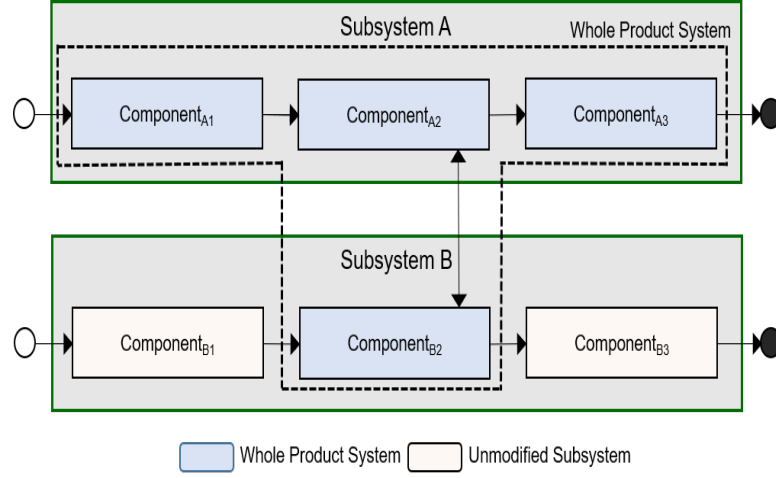


Figure 2.21: Whole Product System Impact for Components in Series. Extended from 2.19 & 2.20

Runtime execution for the whole product system could result in the same underlying components used to integrate the complete process chain to meet the system’s total intended value. Examples of this are data caching mechanisms to reduce the number of queries a database application can handle. Ultimately, both a parallel-in-boundary, and parallel-out-of-boundary could support well-defined functional boundaries for a series of processes executed within a component, subsystem, and finally, the top-level hierarchical system node.

A configuration applies changes to the running system and can cause unexpected behaviours if not specified with the requisite decision impacts. Additionally, the execution of continuous re-configuration could result in an end-state configuration settling at an unintended position. This is referenced earlier as “architectural drift”, Rychly (2014). Petrie (2012) solve for this issue using a concept termed “Backtracking”. This process solves for the last known correct configuration against which the system can validate.

2.2.3.2 System Configuration Capabilities

Extensive IT system deployments naturally contain the notion of configurability, configuration, and configuration management. In that case, dealing with configuration in an IT sense is a conventional systems management function. Configuration, however, in the IoT sphere has gained a fair amount of interest in the research community. This interest is due to the optionality and variability in the configured object domain and how control states occur in devices with minimal onboard resources. Along with protocols for identification, discovery, device, and data management such as those seen in Postscapes (2017), other scenarios are

also present in IoT research. Scenarios that handle; IoT device and service discovery is seen in Datta et al. (2015), Barnaghi and Sheth (2016)) and Ashraf et al. (2016), IoT network management is shown in Bera et al. (2016), and IoT configuration is seen in Perera et al. (2014) and Ashraf et al. (2016). Additionally, the configuration of devices for discovery - using standard architecture components - is also covered Cirani et al. (2014) and Lee et al. (2016). Lee et al. (2016) specifically covers Domain Name System (DNS) auto-configuration for IoT devices. The configuration space for known IoT scenarios are thus bound not only to protocol-type compositions but also actualisation of applied IoT scenarios.

Figure 2.22 decomposes the conceptual overview of the thesis outlined in Chapter 1 by including a general system's configuration capability set. These capabilities support the broad classes of configuration capabilities - such as those found in general configuration management and control systems, Dart (1991b). Dart (1991b) produces definitions, specifications and composition aspects of configuration. These core capabilities categorise more detailed functional capabilities for configuration management and the concepts to produce a configuration management system. Dart outlines the principle functional characteristics based on a standard IEEE viewpoint of configuration, Dart (1991a). The capabilities mentioned in

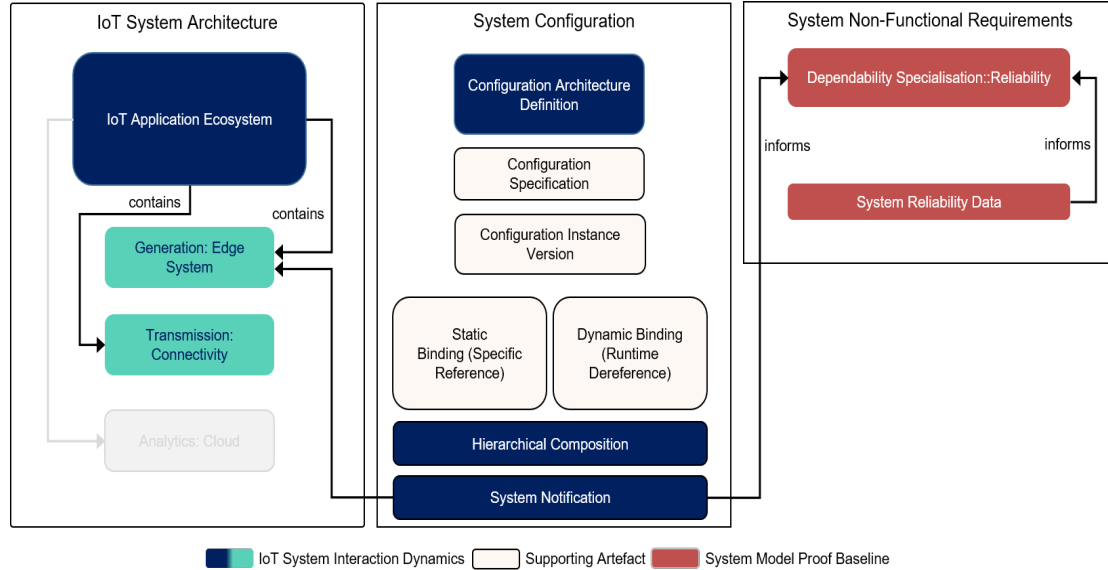


Figure 2.22: System Form Overview - System Configuration Requirements adapted from Dart (1991b)

Figure 2.22 forms part of a foundation of configuration management. However, the study of the runtime configuration for IoT-Ce does not aim to produce a configuration management system. The basis of a runtime configuration expects the existence of configuration item versions, version rollback, auditing, and a full spectrum of management capabilities that

could form part of a configuration management system. Thus, configuration capabilities shall be modelled as purely environmental, having the IoT-Ce system’s primary processes consume outputs of the configuration system. The proposed capabilities in Dart (1991a) that could have an output impact are; Structure, Components, Construction, and Control. Other capabilities such as Auditing, Accounting, Process, and Team functions as regarded are pure configuration management functions.

Configuration management system functions also underpin adaptive systems design based on control systems theory, Cheng (2008). The basis for a state-of-the-art self-adaptation is evident in IBM’s MAPE Framework, IBM (2005) - Monitor, Analyse, Plan, and Execute. In his view on the implementation of RAINBOW, a MAPE-based framework, Cheng implements a more action-oriented framework for an adaptation cycle, namely Monitor, Detect, Decision, and Action, Cheng (2008). Implementation of these requirements can be stipulated by considering other candidate frameworks for analysis—requirements in these frameworks scale from solution-specific requirements to broader industry requirements. The latter is usually a set of known functional requirements that may evolve out of industry engagement. This type of behaviour shows in the delivery of standards by ESTI and the IEEE, and other industry-specific bodies such as the 3GPP. Standards that play a role in the adaptive configurability of systems appears in the table below. Table 2.5 outlines the basis

Table 2.5: Standards, Frameworks and Guidelines for Configuration-based Systems

Description
ETSI GSM Network Configuration Management (1996) for digital cellular telecommunications systems (Phase 2) indicating GSM Network configuration management
IEEE Std 1532-2001 (2001) as the standard for In-system configuration of programmable devices
IBM MAPE-K (2003) as a systems based modelling framework for handling action based goal oriented design, Villegas et al. (2017) and IBM (2005)
Feedback Loop Model of Control Theory (2004) as a standard control theory concept that underpins adaptive systems, Villegas et al. (2017)
Autonomic Computing Reference Architecture (2006) as a multi-layered MAPE-K implementation
Kramer and Magee’s Self-Management Reference Model (2007) as a layered systems architecture for handling goal, change and control requirements, Villegas et al. (2017)
DYNAMICO Reference Model (2013) as a dynamic 3-layered systems architecture for handling adaptation, monitoring and objective control requirements, Villegas et al. (2017)
REFRACT (2014) as a framework for failure avoidance techniques. Extended from RAINBOW - a self-adaptive configuration system, Swanson et al. (2014)

of configurability as the output for concepts such as resource establishment, resource control, failure avoidance, systems adaptivity, self-* system properties, and general systems control. The extension of Table 2.5 by appending additional standards enables single system purpose views that can further argue object, process, and state requirements as pre-conditions for modelling a reduction in complexity of configuration management systems. The thesis does not aim to establish a simplification of modelling a configuration system as these systems

are primarily environmental in the thesis scope.

The MAPE-K Framework used in Cheng (2008) and further in Swanson et al. (2014) is reproduced in Figure 2.23.

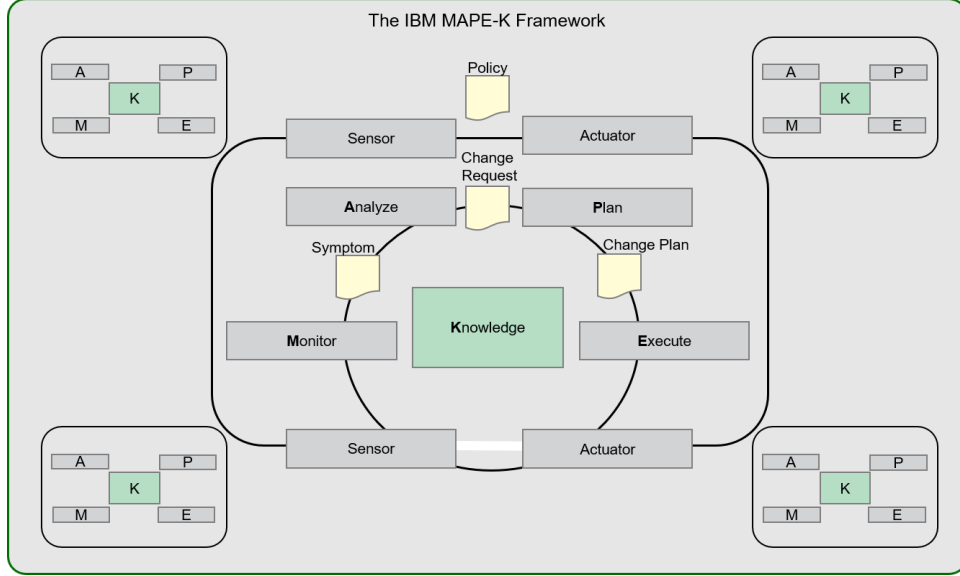


Figure 2.23: Functional Details of the IBM MAPE Autonomic Manager IBM (2005)

MAPE-K supports a degree of the federation of control, asynchronous adaptation interaction, support for failures, and closed-loop feedback control. This thesis refers to the variability of MAPE-K in applying to adaptive systems such as in Cheng’s analysis of MAPE-K. Cheng defines the MAPE-K aligned definitions as;

1. the **Monitor** phase refers to the communication of state from “managed elements”, Cheng (2008). The enablers of this process are communication channels, processes, or any object that contains properties that are useful to determine an object’s state.
2. the **Detect** phase in Cheng (2008) is modelled after the MAPE-K Analyze phase. This phase determines if the system is operating outside of agreed bounds. These could be slow system response, last state update, and so forth.
3. the **Decision** phase in Cheng (2008) is modelled after the MAPE-K Analyze phase. This phase determines the compensating actions that are required to bring the system within acceptable operating ranges.
4. the **Action** phase in Cheng (2008) is modelled after the MAPE-K Execute phase. This phase executes the corrective actions to bring the systems within the desired range.

5. the **Knowledge** repository phase contains the implicit knowledge of the system that can be applied to bring the system back into an operating range. Knowledge types included in IBM (2005) are; Solution Topology Knowledge describing components and configurations, Policy Knowledge describing the limitations and constraints of the system, and Problem Determination Knowledge. These types can describe the functional boundary for a structural and behavioural configuration capability set.

Furthering the extension to RAINBOW, is Swanson et al. (2014) who introduce REFRACT. A framework for failure avoidance techniques in software. The primary function of this system is to manage the avoidance of failures in software from being unmanaged to managed. The author defines self-adaptation as the mechanism that allows a system to continue functioning when the system environment is modified. Within REFRACT, this is a reference to the software configuration. Where these frameworks implement self-healing capabilities, they underpin the need for the existence of a configurable system containing Managers and Execution components. This approach appears aligned to the Backtracking requirement indicated in Petrie (2012), which solves for the last known correct state of a configuration. If this state appears as the correct state, then the Predict and Execute phases for rollback should determine the last appropriate functional state. In the case of REFRACT, this action is termed “Reversion of Configuration”, Swanson et al. (2014). A selection of the reversion is a precursor to reverting the configuration. Hence, the configuration exists before the reversion. Thus, object existence before process existence is argued for in Dori (2002). In REFRACT the underpinning requirement for self-healing is, in fact, a configurable web browser. A software architecture decision - in this case - has guided the fact that something exists before a self-* property can exist.

These adaptation frameworks, and several others, such as those found in more recent research by Villegas et al. (2017), require architectural decisions that underpin selection options for further configuration states due to the dynamic nature of self-healing systems. Design and runtime characteristics of these self-managing systems show that runtime insertion of decisions into runtime execution differs between design and runtime architectures, Anthony et al. (2008). In their view on the ability to configure, Anthony et al. (2008) indicate a need to understand the environment against which runtime processes will execute. This approach deems the need to consider design time splits between static and dynamic parts of the architecture. Finally, framework characteristics that underpin a system model for distributed services indicate that asynchronous service components can deliver highly available and reliable systems, Karamanolis and Magee (1995). The existence of replication servers (Object), server replicating (Process), and server synchronising of replica states (Function)

is used to position a series of functional requirements for system availability. Thus, the characteristics that support adaptive behaviour appear to apply pre-existence principles such as those applied by Dori (2002).

A set of core considerations that reflect as primary drivers for self-adaptation appear in Villegas et al. (2017). These are;

- **Goal.** A goal determines the primary reason for the self-adaptation requirement. Such as; Self-configuring, Self-healing, Self-optimisation, or Self-protection.
- **Structure.** The structure determines the boundary within the system. Namely, is the Adaptation Controller systemic or environmental? In direct control cases, the controller would need to exist as an instrument within the system boundary. Where control is a secondary function, the controller could appear as an environmental subsystem over which the main system has no OPM effect links. For example, a feed-forward controller that does not enable the creation of a closed-loop control system.
- **Reference Inputs.** These inputs create the baseline reference inputs against which a control action is measured, managed by the self-adaptation system. Reference inputs appear as several patterns, such as measurable values, policies, contracts, or constraint states, Villegas et al. (2017).
- **Measured Outputs.** These outputs create the baseline outputs against which a control action executes. These output measures range from discrete values or states against which the adaptation goal compares. Villegas et al. (2017) proposes a case of a dependability attribute, namely availability.
- **Computed Control Action.** These actions specify the action taken to meet the adaptive goals set out in the system. Examples of these contain components that affect behavioural, computing, and architectural aspects of the system, Villegas et al. (2017).
- **Observable Adaptation Properties.** These relay the properties against which a controller can be quality tested. The controller subsystem exhibits stability, accuracy, settling-time, small-overshoot, robustness, termination, consistency, scalability, and security, Villegas et al. (2017). However, the system exhibits properties such as functional invariants, dependability, and security.

The adaptivity concepts mentioned earlier have pre-existing conditions requiring either a physical or informatical object. Thus, the understanding of the existence of an object, or

lack of existence thereof, is a requirement for a system’s adaptivity. A view of the configuration items and their change of state is therefore necessary. In so far as the system’s ability to adapt to changing conditions, without core system capabilities for adaptivity this adaptation cannot be realised. For example, in cases of computed control actions, a set of discrete operations can affect the functions of a system, Villegas et al. (2017)(pp.26). Thus in OPM ISO-19450 terms, **Discrete Operations** affect the **Managed System Software Architecture Set**. Additionally, in the case of observable adaptation properties, a controller can exhibit some form of observable adaptation characteristic that defines the managed system’s reliability, Villegas et al. (2017)(pp.26). For the considerations of Goals—to—Properties in self-adaptive systems, the existence of an object or process is necessary, hence a system architecture, and therefore variation in system configuration through an understanding of the system’s capabilities.

The need for context awareness, adaptation, and object intelligence appears as research goals in the earlier Table 2.1 by (Vermesan et al. (2009)). Additionally, the view on Self-* properties was also present as a research goal in research positions such as Ailisto et al. (2011) and Miorandi et al. (2012). Underpinned by the research goals to position device and standards growth back into the IoT market, a need for understanding the configuration of IoT was a key driver mentioned in a 2017-2018 technology research statement by Gartner (2017). This configuration statement was related to the management of things that required a review of IoT monitoring and management at scale. Capabilities exist today that enable virtual state management of connected-edge devices. “Device Shadowing” is a feature available on cloud-based IoT platforms such as Amazon Web Services, Varga et al. (2018). Similarly, the notion of autonomous things intensifies due to the forecast of interactions between billions of smart devices. Ailisto et al. (2011) lists these properties as; self-configuring, self-protecting, self-organising, self-optimising, self-reliant, self-healing, self-aware, self-learning, self-adjusting, and so forth. Both sets of research agendas indicate the need for an architecture; however, the systems viewpoints on modelling systems are focused on interaction standards of physical devices but not on the conceptualisation, and true systems architecture of the system as a whole. At the time, perhaps this was planned in the Evolutionary Standards section of the research direction, by Vermesan et al. (2009). However, by 2009, OPM modelling was already available for more than five years. It is only five years after the Strategic Research Agenda for IoT, Vermesan et al. (2009), that OPM reached ISO standardisation, ISO/TC 184/SC 5 (E) (2014). By 2011, the definition of IoT was updated to include the words “conceptually defined” by the authors of the Strategic Research Agenda for IoT, Vermesan et al. (2011). Although the capability of context

awareness, adaptation, and object intelligence was noted in 2009, the modelling of these capabilities can now use a method that can integrate cross-functional modelling efforts while meeting model simplification goals.

The primary approach viewpoint on Runtime Configuration in Chapter 1 outlines;

1. the *ability to view the arrangement of components and subsystems in the IoT-Ce as hierarchical* in nature,
2. *autonomic frameworks which expect the existence of a system* that requires corrective control, and
3. *viewing the output of configuration management as systemic*, while viewing configuration management processes as environmental in the IoT-Ce.

The Runtime Configuration background in Chapter 2 produces further appreciation for the concepts in Chapter 1 as;

- the *notion of a variable arrangement of system, subsystem, and components* that exist in a configuration space,
- the *conceptual abstraction of configurability*, configuration, and configuration management functions,
- the *unique view on configuration as a pre-existence criterion* for re-configuration,
- the *runtime component arrangement of single or multiple function blocks* that aid in hierarchical system design and system placement, and
- a view on the *systemic and environmental component requirement possibilities* that support adaptive systems design.

2.2.4 Architecture Configuration Decisions

Arranging architectural decisions into a structured qualitative decision set occurs using scenario modelling, Crawley et al. (2016). As indicated in the Crawley et al. (2016) view on architecture decisions, architectural principles guide the systems decision-making process. These concepts focus on creating an architectural decision space, measurable indicators, and segregation of decisions between systems and other decisions. Additionally, by creating discrete, measurable decision parameters, a system can be analysed qualitatively as each entity and relation can contain variable strength indicators for each relationship and scenario. This qualitative evaluation is also a known approach when considering tradespace tradeoffs

Crawley et al. (2016).

In order to concur with the view that “A single model should never be the only V&V tool, but should show how the model is used to validate a process or product”, Cameron et al. (2017). A secondary model, apart from those generated out of OPM, is used to outline the variations of decisions for a system. This decision model realises through the use of Design Structure Matrices (DSMs) outlined in Eppinger and Browning (2012), and supported by the views indicated in Crawley’s chapters on *Architecture as Decisions*, Crawley et al. (2016). Eppinger outlines the advantages of using DSMs as;

1. producing *a compact representation of structure and behaviour* of a system,
2. *enhancing visualisation* to enable global system decisions,
3. *reducing of complexity* as hierarchies in the system become evident,
4. providing *the basis of mathematical analysis*. DSM analysis can also provide insights into unknown links, component impact propagation, process loops, modularity, and other important patterns and effects, and
5. handling model flexibility to extend use to other data analysis opportunities.

Additionally, the primary characteristics of DSMs indicated by Eppinger and Browning (2012) are shown to;

1. provide *emphasis on interactions*,
2. outline *structure to interaction requirements*; Sequential, Parallel, Coupled, Conditional,
3. enable *decision iterations, interaction strengths, and minimising impact focus on boundary conditions*,
4. enable *flow sequencing for data, system decomposition*, and convention agreements. These are applicable to Input Column notation only,
5. indicate *coupled activities*, and
6. support Path Searching, Basic Block Sequencing, and Powers of Adjacency algorithms.

A model that structures the scope of architectural decisions becomes quite large if one considers the vast body of IoT-Ce knowledge discussed in preceding chapters and sections. On the use of DSMs in Eppinger and Browning (2012), the constructs of DSM drive the

principle of compact and concise representation of components, processes, and interacting activities. Therefore, using an unbounded body of research yields a comparable unbounded DSM that does not deliver the requisite clarity in the system specification. Similarly, this does not yield an increase in the credibility of the model output. Eppinger and Browning (2012) indicates the use of four DSM types that support variations of component design, organizational design, activity design, or a multi-mode design that contains combinations of these DSM types. Research in structured matrix design for processes using OPM is already an area of study, as seen in Sharon et al. (2012) for structured project delivery, and more recently in Sobol and Dori (2020) for OPM thing flattening. These research views investigate OPM for use in structured project delivery processes and the flattening of hierarchical thing structures. Where conventional usage focuses on method-specific mechanisms between OPM and DSM, the use of DSM in IoT-Ce architecture decision trade-off produces uniqueness in the ability to;

- identify decision impacts for diverse parameters sets,
- identify decision clusters due to high interdependence between decision parameters, and
- apply declarative decision requirements for concepts such as state-awareness.

A simplification of the DSM uses a series of process inputs and outputs. Figure 2.24 shows a simplified transition extracted from DSMWeb (2019) however, a similar transition set is referenced in Sharon et al. (2012), and Bradshaw et al. (2012). Figure 2.24 uses input column view, which differs to the dependency view created by Bradshaw et al. (2012). In the task

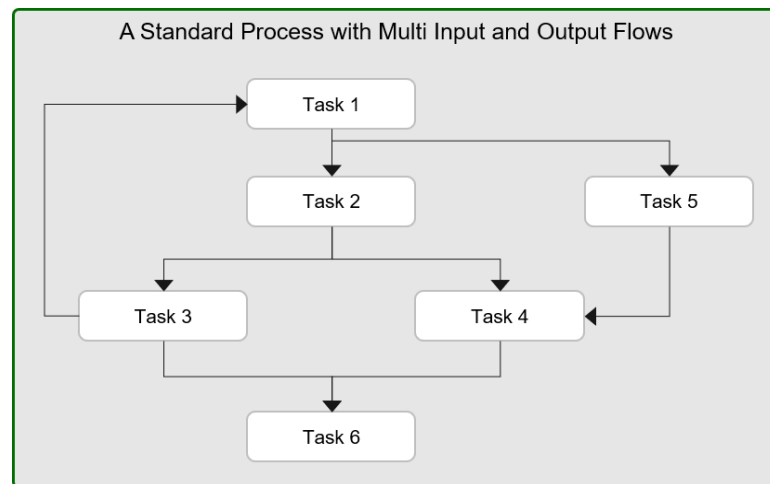


Figure 2.24: A Simple Process to DSM Summary extracted from DSMWeb (2019)

flow in Figure 2.24, the following is applicable; Task 1 sends outputs to both Task 2 and Task

5. These will be inputs received for Task 2 and Task 5. Therefore the cells that intersect task row 1 and columns 2 and 5 are marked to indicate an interaction between these tasks. Task 1 receives input from Task 3. Task 2 sends output to both Task 3 and Task 4. Task 3 sends output to Task 1, and Task 4 receives input from Task 5. Both Task 3 and Task 4 sends an output to Task 6. As indicated by DSMWeb (2019), the intersection between row and column in the DSM can contain either relational indicators or strength indicators. The input column approach views the row as the output and the column as the input receiver. This flexibility makes the DSM, or extension of the DSM, a viable model type for checking automated system testing scenarios and architectural decision dependency analysis. Table 2.6 shows a well-known tabular view of the DSM, that applies a simple boolean output of Figure 2.24. Crawley et al. (2016)(pp.336) indicates DSM as supporting representative and structural

Table 2.6: A Simple DSM Process Equivalent extracted from DSMWeb (2019)

Task	1	2	3	4	5	6
1		x			x	
2			x	x		
3	x					x
4						x
5				x		
6						

layers in a decision support system, but not a pure simulation or graphical representation of evaluation results. The DSM is, however, well positioned to outline different configurations and the associated strengths between architectures. DSMs, and their counterpart OPM outputs, play a key role in systems engineering in that it enables the modelling of a systems view of the world, Crawley et al. (2016). Underpinning architectural options analysis using DSM, the use of architectural principles guide the decision-making process, Crawley et al. (2016)(pp.326-327). These architectural principles summarise as;

1. setting the *“boundary of the architectural space”*,
2. evaluation of the architecture through the *impact that metrics have on the architecture*, and
3. the *separation of architecture-only decisions* from larger system decisions.

To support the extraction of column and row parameter sets, or simply architecture configuration decisions, IoT design and evaluation methods such as ATAM contain tradeoffs, sensitivity points, and other pertinent system attributes for making sound architectural decisions, Washizaki and Tei (2019). These are used in the evaluation process within the architectural decision boundary to deliver a well-rounded set of decisions for different system configurations. Recent interest in IoT Systems design and evaluation has shown that

a modular approach to decisioning is needed to elegantly position systems that adhere to hierarchical constructs, Washizaki and Tei (2019). As seen in the tradeoff analysis for this approach, ATAM enables creating attribute-based concepts that offset to create different architectures, Kazman et al. (1998). ATAM is a circular decision method characterised by the following steps;

1. Collect Scenarios,
2. Obtain Attribute-based Requirements, Constraints, and Environment,
3. Generate Candidate Architectures within Design Space,
4. Generate Attribute Analysis,
5. Identify Attribute Sensitivity, and
6. Identify Attribute Tradeoffs.

Trade-off sensitivities can appear naturally in n-squared matrices such as DSMs. The natural fit of row and column intersections enables interaction presence and variability of interaction strengths later in this thesis.

The primary approach viewpoint on Architecture Configuration Decisions in Chapter 1 outlines;

1. the *ability to extract a viable parameter set that aptly describes configuration decisions* for enrollment of smart mobile devices in an IoT-Ce context,
2. the *ability to apply standard Python scripting libraries* in order to simulate model validation, and
3. the *ability to apply a series of accepted constraints and control* against a historical failure data set.

The Architecture Configuration Decisions background in Chapter 2 produces further appreciation for certain concepts in Chapter 1 as;

- a position on the use of a *structured secondary model to present IoT-Ce architecture decisions*. The use of a DSM not only creates a concise view of the entire IoT-Ce systems architecture but also exposes its use to a simulation-ready proof for the converged model of the vertical themes positioned in Chapter 2, and

- an approach for *dealing with DSM model content using sensitivities of IoT-Ce architecture decision parameters*.

The ISARC model proof appears in detail in Chapter 5. A similar series of background characteristics on dependability, reliability, and random system behaviour underpins the Python codebase that produces the secondary contribution covered in this thesis. This background appears in Appendix A. The secondary contribution in the thesis provides a method to programmatically validate the ISARC using OPM parameter extraction and Design Structure Matrices (DSM). The method for modelling configuration decisions for the IoT-Ce uses OPM as the basis for parameter extraction and DSM to represent the architecture. A set of suitably bound decision parameters supports a test of conceptual architecture variations of the ISARC using a novel concrete control design based on instantaneous and accumulated errors. This contribution's primary benefit extends the systems architecture by proving that the ISARC can support ongoing runtime configuration requirements using a series of simulated data set samples using Python. A series of data generation code blocks output the result of interaction decisions between ISARC model components. These outputs compare a historical data set that mimics a hierarchical systems architecture found in a failure trace data set for a multi-tiered DNS.

2.3 Chapter Summary

The concept of a connected world consisting of connected things communicating with each other is not new. However, the concept has evolved with the connectivity to televisions, mobile phones, air conditioners, and other appliances regarded as the next wave of actual control and automation.

An IoT systems architecture positions the need for a broader appreciation of the requirement to configure systems developed by a varying number of device and software manufacturers. An arrangement of subsystems and components is maintained while these systems are in operation in the known and unknown spectrum of human needs. The problem statement outlines as trying to understand the role of this arrangement and possible changes to it in the face of complex data flows within a constrained IoT systems environment. For this to be accomplished, there is a need to abstract the architecture from the noise caused by unsure beneficiary value in IoT. Unclear system decisions primarily cause this noise.

This chapter deals with the background of several unique concepts that exist at different levels of maturity. These concepts have developed independently over a long period.

Systems Architecture and Systems are known to be driven by composition and hierarchy since the mid-1950s. *Structured Decision Matrices* are concepts born in the 1960s. *IoT* is a relatively new concept spurred by an influx of edge device capabilities since the late-1990s. Thus, the specification of a systems architecture for *IoT-Ce* is a unique convergence of these concepts. The resultant system architecture and architecture decision method is supported by the OPM ISO-19450 - an ISO standardization that changes the importance of objects, processes, states and model-to-text dual-modality.

The proposed IoT-Ce systems architecture and decision model is necessary to provide the motivation needed to support claims that propose measurable impacts of IoT system architecture variations on the reliability of edge device state notifications in single subsystems or multi-node interconnected systems.

Chapter 3

Systems Architectures and IoT-Ce Configuration

“A symbol system —a language—that can conceptually model a given system using ontology with fewer diagram kinds and fewer symbols and relations among them is preferable to a larger language with more diagram kinds and more symbols and relations among them.” The Minimal Conceptual Modelling Language OPM Principle - Professor Dov Dori (2016a).

This chapter deals with the direct requirements for the IoT-Ce systems architecture by realising these requirements using ISO OPM-19450. Systems architecture is defined as the *“embodiment of concept, the allocation of physical/informational function to the elements of form, and the definition of relationships among the elements and with the surrounding context.”*, Crawley et al. (2016)(pp.124). The key guidance on creating a systems architecture is mapping instruments that deliver capabilities to the internal processes of the IoT-Ce system, subsystems, or components. Secondary systems that provide supporting capabilities to the IoT-Ce system require identification as the build of these systems can impact how much process integration needs to occur. IoT-Ce system architectures realize through the structure, behaviour, and interactions that provide the value-based output of the IoT-Ce. The IoT-Ce system itself has a purpose, primary function, agents or instruments, an environment, and a problem occurrence. The structure is characterised through the physical system; its behaviours define its ability to create value through the system structure. The structure enables the execution of processes. Chapter 3 introduces the IoT Systems Architecture for Runtime Configuration (ISARC). The ISARC outlines the requirements definition of the system by implementing a partial conformant OPM ISO 19450 design. A set of filtered requirements extract from the primary themes identified in preceding chapters. Chapter 3 creates an integrated vertical-thematic view of system requirements using objects, processes,

and states. A pure theoretical view of OPM applies the notion of an object as an entity, a process as a method to change the object's state, and the state as the sum of activities. This approach sets the initial view of dealing with variations in the architecture once the system actualises. These variations underpin the architectural decisions needed to support qualitative system testing using independent system attributes.

Finally, Chapter 3 produces an outline containing the requirements to deliver state-aware reliability within the IoT-Ce for Android device states. Chapter 3 proposes the use of standard OPM naming to produce a set of self-standing decision parameters to elevate discussions on concepts of; object essence, whether physical or informatical, object placement, whether systemic or environmental, and pre-existence for ordered object transformation.

3.1 ISARC Requirements Extraction

The ISARC creates a standards-based model that extracts requirements from several literature sources referenced up to this point in the thesis. Conformance to the OPM ISO-19450 standard simplifies the extraction of requirements into three key types; objects, processes, and states. As previously indicated, OPM in systems architecture exhibits three conformance levels; partial, full, and conformance by toolmakers. Each group of conformance clauses in ISO/TC 184/SC 5 (E) (2014) guides the level of alignment to OPM through the use of specific symbols and modelling requirements. The conformance approach that guides the ISARC is based on modelling parts of an existing body of knowledge to propose a systems architecture for the IoT-Ce. Thus, a partial conformance profile is sufficient to represent the system diagrams (SDs) of interest in the primary system. In stating the systems architecture and subsequent decision model, partial conformance requirements to ISO/TC 184/SC 5 (E) (2014) apply as;

1. symbolic conformance to *objects*, *processes*, and *states*. Partial conformance requires adherence to the definition and use of objects, processes, and states in conformance with the constraints placed on OPM Things. Such as object pre-existence as the core requirement for process transformation. An object can exist before a process transformation occurs, but a process cannot exist without an object ISO/TC 184/SC 5 (E) (2014)(Section 7 OPM Thing Syntax and Semantics)
2. symbolic conformance to OPM syntax and semantics in cases of *procedural link usage* ISO/TC 184/SC 5 (E) (2014)(Section 8 OPM link syntax and semantics overview),

3. symbolic conformance to all other link types such as *transformation links*, *consumption links*, *enabling links*, *control links* and *condition links* ISO/TC 184/SC 5 (E) (2014)(Section 9 Procedural links),
4. symbolic conformance to *structural links and cardinalities* ISO/TC 184/SC 5 (E) (2014)(Section 10 Structural links & Section 11 Relationship cardinalities) and
5. symbolic conformance to *logical operators* ISO/TC 184/SC 5 (E) (2014)(Section 12 Logical operators: AND, XOR, and OR).

OPM symbols represent structural and interaction types to model systems, Dori (2011)(p.221). These symbol types support any combination of interactions such that it produces a single set of similarly structured Object Process Diagrams (OPD). OPM research and tools



Figure 3.1: OPM Symbol Toolbar. Extracted from OPCAT 4.2 Toolbar

indicate three clusters of symbols. The three cluster layout from the OPCAT 4.2 toolbar produces a simple reference. The clusters on the toolbar contains;

- **Objects, States and Processes** in Cluster 1. For IoT-Ce, examples such as **IoT System Set**, **Front-end System Set**, **IoT Configuration Option** with states **Unknown** and **Known**, and **Device Communicating** can be represented, respectively. As noted in preceding sections on OPM, **objects** appear in green, while **processes** appear in blue.
- **Structural symbols** in Cluster 2, which include Aggregation-Participation, Exhibition-Characterization, Generalization-Specialization, Classification-Instantiation, Unidirectional, and Bidirectional Relations, and
- **Procedural links** in Cluster 3, which includes an Agent Link, Instrument Link, Result or Consumption Link, Effect Link, Instrument Event Link, Consumption Event Link, Condition Link, Exception Link, and Invocation Link. A procedural link denotes how the system operates to achieve its function using the entities linked by the procedural relationship, ISO/TC 184/SC 5 (E) (2014)(pp.13).

Table 3.1 contains Object Process Language (OPL) equivalence samples for IoT-Ce model components. The examples used in Table 3.1 aims to propose a simplification of possible complexities relating to the fit of OPM, and by extension, the ISARC, for specifying subsystems and components in the IoT-Ce. The OPM Metamodel in ISO/TC 184/SC 5

Table 3.1: OPM Toolbar Usage Sample

Cluster	ID:OPM Symbol	IoT-CE OPL
1	1:Object	Front-end System Set, Near-end System Set, Controller System Set
1	2:States	Device State::Connected or Not Connected, Device State::Sleep or Standby or Active
1	3:Processes	Device State Notifying, Gateway Attach Denying, Gateway Communication Failing
2	1:Aggregation-Participation	IoT System Set consists of Front-end System Set, Near-end System Set, and Far-end System Set. Front-end System Set consists of Smartphone Devices.
2	2:Exhibition-Characterization	Device exhibits Status
2	3:Generalization-Specialization	Reliability is a type of Dependability.
2	4:Classification-Instantiation	Android is an Operating System.
2	5:Unidirectional Relation	Status relates to Operating System
2	6:Bidirectional Relation	Operating System consists of Runtime Code Components. Application System consists of Runtime Code Components. Shared Library Code base and Runtime Code Components are (possibly) equivalent.
3	1:Agent Link	End User is physical. End User handles Front-end System Set.
3	2:Instrument Link	Self-adaptation IoT System Selecting requires Configuration System Set, Front-end System Set, and IoT Protocol Set.
3	3:Result/Consumption Link	Self-adaptation IoT System Selecting consumes IoT System Set.
3	4:Effect Link	Configuration Option Selecting affects Configuration System Set.
3	5:Instrument Event Link	Variable Configuration Options triggers Option Selecting when it enters State Unknown.
3	6:Consumption Event Link	Variable Configuration Options triggers Self-adaptation IoT System Selecting when its state changes.
3	7:Condition Link	Self-adaptation IoT System Selecting occurs if Front-end System Set is in existent either Near-end System Set is in existent, or Front-end System Set is in existent.
3	8:Exception Link	Option Selecting triggers Self-adaptation IoT System Selecting when it lasts more than 5 seconds.
3	9:Invocation Link	Self-adaptation IoT System Selecting invokes Option Selecting.

(E) (2014)(pp.15) contains several core definition classes that outline abstract viewpoints such as *Things*, to more concrete concepts such as transformation of objects by [processes](#). Jointly, the combination of these elements produces the constructs of the actual systems architecture. Furthermore, OPM model representations such as those seen in Mordecai and Dori (2017a) propose a bridge to a gap in cyber-physical systems today. In Dori’s view on model constructs, OPM model elements represent a purpose-built, “compact” set of elements, Mordecai and Dori (2017a). These model elements represent instances of the meta-model overlaid with the basic form-function model shown in Figure 3.2.

Figure 3.2 outlines the functional and structural constructs of the OPM. Crawley et al. (2016)(pp.98) summarises a systems model containing a function consisting of a process and

an operand against which the process executes. The function attributes to the creation of the performance of the system. Additionally, an instrument enables the process execution. As shown in Figure 3.2, the operand contains the actual state, which changes once a process enacts on it. A final description of the model elements for cyber-physical systems shown in

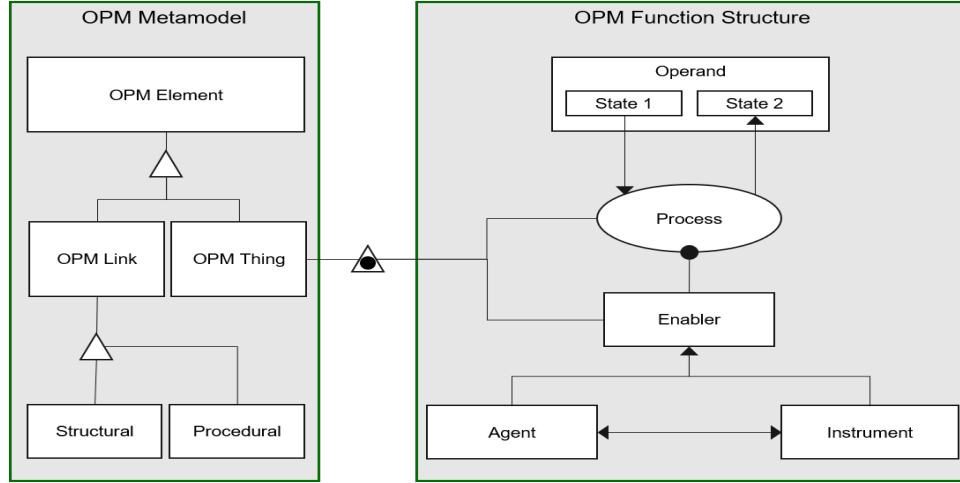


Figure 3.2: OPM Function Structure Overview. Adapted from ISO/TC 184/SC 5 (E) (2014)(pp.13) and Crawley et al. (2016)(pp.101)

Mordecai and Dori (2017a), indicates the model as having;

- *Things* that are either physical or informatical containing attributes that are either systemic or environmental, and
- containing links showing structural or procedural constructs.

Figure 3.2 outlines the following definitions;

- The OPM Metamodel contains elements, of which links and *Things* are specialisations. Structural and procedural links are specialised OPM links,
- an OPM Function-Structure has an operand that contains states that undergo change by means of a process. The process is handled by an enabler that is either a human agent or informatical instrument - essentially, of physical or informational essence, and
- the OPM Function-Structure is an instance of the OPM Metamodel.

The requirements model extracted from the large body of knowledge from Chapter 1 and 2 is adjusted via an OPM filter which implements OPM template design—the what and how in systems architecture design—as positioned in Soderborg et al. (2003). Figure 3.3, presents the top layer body of knowledge, the standards that govern the assignment of requirement, and the layer that yields the object, process, and state outputs. Crawley et al. (2016)(pp.63)

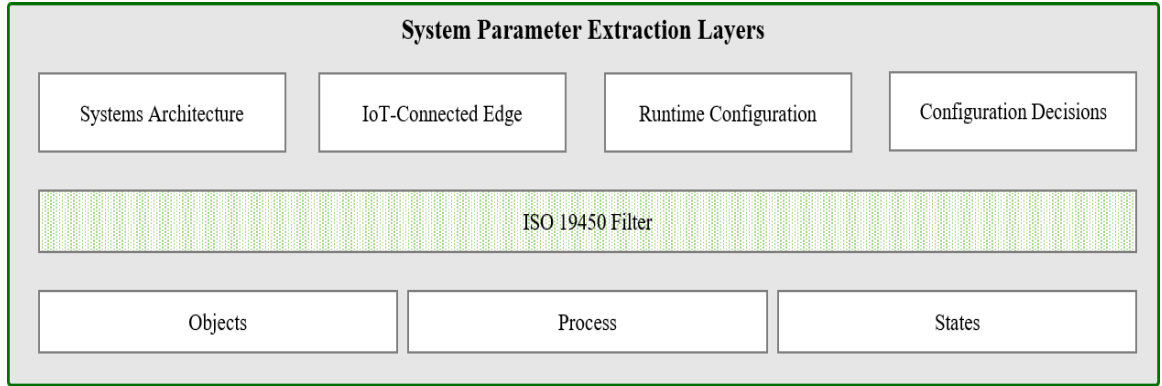


Figure 3.3: System Requirements Extraction Layers. Adapted from Crawley et al. (2016)(pp.63) and ISO/TC 184/SC 5 (E) (2014)

indicates that the approach for dealing with complex systems derives from decomposition, hierarchy modelling, specialisation and generalisation, detailed iteration, and finally, view construction as visual output. The output of the extracted requirements yields multiple IoT-Ce decision parameters.

3.1.1 Additional Conformance Requirements

During the process of producing a systems architecture, additional conformance requirements can ensure that secondary subsystems and components are present in the system model. These ancillary systems aids in delivering primary system value by having an either systemic or environmental impact. In the case of the ISARC, objects such as network and data protocols can either serve as part of the primary system or impact the main system value without an environmental object or process being affected. Examples of these protocols are seen in the classification of protocols in the earlier IoT protocol section that delivers addressing, discovery, and management functions in Figure 1.2.

The argument here is that not all protocols are necessary for the delivery IoT-Ce system beneficiary value. One could go as far as claiming that no conventional IoT protocols are needed to execute the data acquisition process for a smartphone that requires only using on-board sensing capabilities. Furthermore, in the fragmented sense, it would not be reasonable to consider a single protocol for application-level data transmission, as the computational requirements between **Front-end System Set**, and other IoT ecosystem architectures differ based on the system purpose for each domain. This view is evident in the changing requirements between web and IoT protocols represented in Figure 2.14. Simply put, the change of identification, discovery and management protocols from classical communication towards

lossy device communication favours a different set of fundamental system goals. The optimised selection of protocols based on **Front-end System Set** capability, IoT solution set, or IoT product varies due to functional and non-functional requirements. Therefore, the additional conformance is based not only on the modelling symbols in a partial conformance OPM model but also on the application of requirements that could impact the IoT-Ce's runtime configuration. Thus, the additional conformance requirements argue for in-system boundary or outside of system boundary considerations.

Additional conformance for deciding state awareness is handled in OPM using the CPGAME. In OPM for CPS gaps, Mordecai and Dori (2017a) indicate the static and runtime knowledge of the system as the knowledge base and information base of the system, respectively. Using the cyber-physical gap awareness model (CPGAME) by Mordecai and Dori (2017a), their modelling approach sees five important attribute considerations of the model: *entity-type*, *entity-detection in the system*, *feature awareness of detected entity*, *state-space awareness*, and *state-perception of an entity in the system*. When modelled into the IoT-Ce systems architecture, **Front-end System Set** model attributes can be attached using a similar classification. Thus, a proposed mapping adapts the attribute values from the example in Mordecai and Dori (2017a), towards an IoT-Ce description set in Table 3.2. Once integrated, these

Table 3.2: Systemic Conception of the IoT Connected Edge adapted from Mordecai and Dori (2017a)

System Model Awareness	Description
Entity-type	Front-end System Set
Entity detection in the system	Front-end System Set detected in sensing coverage area
Feature awareness of entity	Front-end System Set is aware of connectivity state space
State-space awareness	available, unavailable, unknown, known
State-perception of entity	Front-end System Set -available, Front-end System Set -unavailable, Front-end System Set -unknown, Front-end System Set -known

considerations enable the ISARC to view architectural relationships between concept and validation.

Thus, the considerations for reliable state-awareness in the IoT-Ce over the magnitude of variations of IoT can be bound in a system boundary. This limiting construct occurs due to the limited but varied IoT-Ce solution protocols and mechanisms that enable connectivity for **Front-end System Set** instances. Therefore, abstractions such as hardware abstraction, **Front-end System Set** Operating System abstraction, and configuration abstraction in the systems architecture themes make the model more plausible. Process requirements change

the state of each awareness concept in Table 3.2 from **unknown** to **known**, indicating an explicit awareness of state change in the system.

The conformance requirements outlined in the remainder of this chapter summarises three CPGAME approach steps into the ISARC’s primary and secondary processes. The modelling of entity acquisition, representation management, and interaction analysis proposes to form part of the ISARC knowledge base such that it forms a structural decision construct for each requirement. This approach is proposed, amongst other structured questions, to output a single decision matrix for each primary system function. The resultant start and completion sub-process in each primary process hide the internal complexity of the individual primary processes. Table 3.3 indicates the candidate start and end activities for each part of the awareness process.

Table 3.3: Adapted Cyber Physical Gap Process Interaction. Summarised from Mordecai and Dori (2017a)

Primary Process	Initiating Subprocess	Finalising Subprocess
Entity Acquisition	Entity Detecting	Acquisition Completing
Representation Management	State Acquiring	Process Completing
Interaction Analysis	Behaviour Representing	Impact Recording

The *entity acquisition* processes in Table 3.3 plays a role in determining the pre-existence of an entity in the system. Within the entire system - in-boundary or outside - different architectural decisions increase the combinations of decision cluster inputs and outputs. Therefore, sensitivities for individual decisions are important as this can be used to prioritise decision impacts across non-functional requirements. As seen in the summary of trade-offs in section 2.2.4, scenario collection is a precursor to tasks on attribute identification, parameter identification, and sensitivity identification. In the case of the ISARC, these scenarios lead to those that impact our understanding of the system state at a point in time.

The second process construct is the *representation of state* in the system. The acquisition of state in the system considers the level of impact of the state on the system. In Mordecai and Dori (2017a), these classifications determine the ability to integrate relative evasive measures that need to occur to ensure the system’s stability. The ISARC regards this as a decision that impacts the ability to connect to an ED/GW within the sensing boundary, addressing boundary or pre-existence object boundaries in the OPM. Of course, several environmental factors can cause outright connectivity failures, such as failures caused by **Front-end System Set** Media Access Control (MAC) filtering, **Front-end System Set** black-listing, or system policy enforcement. These factors are regarded as environmental in that

the system cannot control the outcome unless explicitly instructed to do so. Thus, the first connection or failure measures in accordance with systemic factors such as the **IoT Protocol Set**, **Front-end System Set** addressing against a DNS, or user-instructed process terminations.

Finally, the *interaction analysis* process uses interaction and action analysis to determine and map interaction paths that could lead to system failure. The path seen for error to faulted system states is represented in Appendix A, section A.1.2, Figure A.1. In this view, an escalation process controls the outcome of handling recoverable errors towards a faulted system state that is not recoverable. Handling a failed state via the restart of an **Front-end System Set** service is a decision that can tradeoff against holding internal **Front-end System Set** state. The latter will require increased storage and possibly increased onboard processing on the **Front-end System Set**.

In summary, additional considerations are those model guides not caused due to OPM symbol usage but rather the implicit model constructs that determine the fidelity of the system model. In this section, five model considerations in Table 3.2 and three process markers in Table 3.3 guide the proposed fidelity increases in the ISARC decision parameter list. The markers largely create a defined state-awareness handling mechanism, against which architecture decisions validate. The implementation of the structural path to guide efficiencies in architecture decisions based on the ISARC appears in Chapter 5 - Model Validation & Evidence.

3.2 State of the Art Systems Architecture Realisation

Coupling too many systems too tightly into a primary system boundary can cause internal process interleaving and create unintended impacts on the primary system. Decoupling occurs by stating system interfaces at the boundaries between systems. However, the unintended impact of constant coupling and decoupling decisions that cause shifts in systemic and environmental (object, process, and state) concerns can raise the risk of system drift and settlement in an unintended state, as seen in earlier notions on dealing with adaptivity, Rychly (2014). The ISARC meets hierarchical system requirements, such as those found in section 3.1. It meets this by applying strict OPM usage controls. OPM supports converged model representation, boundary thinking, and modular structure with implicit order execution ISO/TC 184/SC 5 (E) (2014). Thus, realising the model intended to provide system value while wading through principles of hierarchy, boundary, model assertion, fidelity, and credibility, the primary model realisation bears five key concepts; *Problem Occurrence*, *Purpose*, *Primary Function*, *Enabler*, and *Environment*. The assertion that these questions

form the basis of sound systems design applies the ISARC inputs in Figure 3.3.

A set of systems architecture actualisation considerations indicate that a set of system definitions apply that each system concept can play singular or multiple roles in the design. The list below extracts from the section on *defining architectures* in Crawley et al. (2016)(pp.126). In defining the architecture, As seen in Figure 3.4, the system architecture is defined as a combination of form and function. In the case of form, a set of entities and relational constructs are evident. These deliver the physical system.

The questions for defining architecture summarise in Crawley et al. (2016) as;

- *Internal processes* which have the necessary instruments to support process execution. This definition yields an IoT-Ce requirement if internal device state management requires additional control to fulfil the primary state notifying process.
- *Additional processes*, instruments, or operands required to fulfil the delivery of value. This definition yields an IoT-Ce requirement if the need for **Stable Connectivity** exists as a secondary process to fulfil state notification.
- *Supporting processes*, instruments, or operands required to fulfil the delivery of value. This definition yields an IoT-Ce requirement if the need for **Data Cache Handling** that supports operational state notification exists.
- *System Boundary Interfacing*. This definition yields an IoT-Ce requirement if device handover interfaces between the **Front-end System Set** and **DNS** addressing nodes exist.
- *Sequenced executions* to produce Primary and Secondary functions. This definition yields an IoT-Ce requirement if **Device State Power Handling** changes device State from **Standby** to **Sleep** after **IoT Gateway Cache Clearing** occurs.
- *Parallel Functions*. This definition yields an IoT-Ce requirement if **Gateway Cache Clearing** and **Gateway Persistent Store Updating** occurs in parallel.
- *Clock Time*. This definition yields an IoT-Ce requirement if component clock synchronising needs to occur.

3.2.1 Architecture Guidance on Form & Function

The structured architectural guidance that grounds the requirements in Chapter 3, and further the outputs in subsequent chapters, references from Crawley et al. (2016)(pp.97-123) as;

- “function is what a system does”,
- “function is equal to process plus operand”,
- “function has entities” and “relationships” that represent interaction,
- “function emerges from functional interactions between entities”,
- “operation, or transformation contributes to performance”,
- “operands are objects, acted upon by the function”, additionally, “they need not exist prior to the execution of function”,
- “process is the creation or destruction of an operand” It is a pattern of transformation undergone by an object,
- “Function and value are always delivered at an interface in the system boundary”,
- “internal function analysis uses reverse engineering of form and standard blueprints”. Internal functions follow a Noun-Verb gerund construct. A noun holds the state and verb executes a process to change the state,
- “functions plus functional interactions represents the functional architecture”,
- understand the “value pathway” by indicating entities of form, which is not part of functional architecture but provides an instrumental role to processes. Additionally, processes and operands can support secondary external functions,
- “shared operands” can be shown using “projections” as it represents simplified interactions, and
- function classifications that exist in general systems theory as per Høyland and Rausand (2009)(pp.78). These functional classifications are; Primary, Auxiliary, Protective, Information, Interface and Superfluous.

As seen in the function definition constructs above, a function appears as the combination of operand and process. An operand manifests as an object that holds the state of an entity

outside of a process. The process is an action that changes the state of an operand. Somewhat different from an instrument, an operand can either exist or not before the function executes. As shown in the function block in Figure 3.4, Process 1 interacts with Object 2 to interact with Process 2. Processes execute in a sequence, enabling a change in the state of operands in a system, Crawley et al. (2016). The consistency of function, form, and architecture descriptions from the preceding chapters 1 and 2 builds a critical mass in placing the correct placement of realisation process and definitions for the ISARC. Thus,

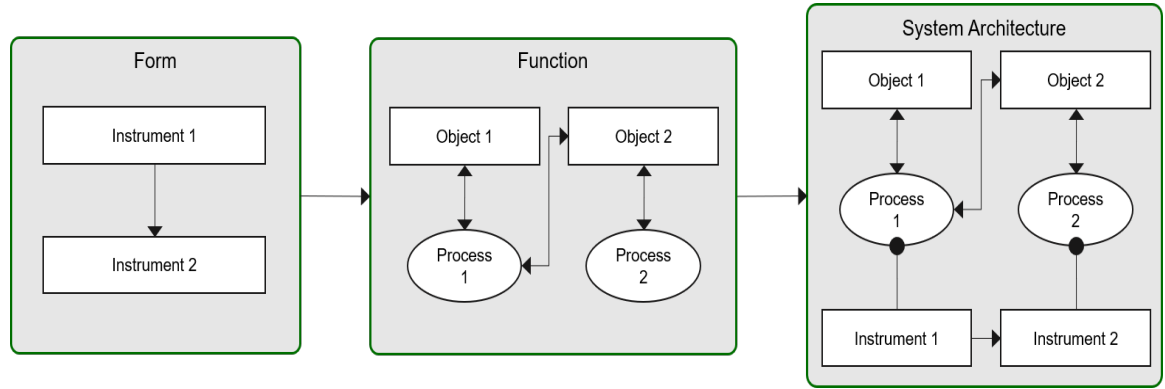


Figure 3.4: Definition of a Systems Architecture. Adapted from Crawley et al. (2016)

the ISARC adheres to a naturally contained definition set and model output compliant to the required OPM symbol usage. Additionally, the identification of the primary function appears in a process adapted from Crawley et al. (2016) and ISO/TC 184/SC 5 (E) (2014).

Figure 3.5, expands the considerations seen in the architecture definition questions in Crawley et al. (2016)(pp.126), the form function interaction as shown in 3.4 and OPM ISO-19450, to propose a thesis specific iterative and cyclical decisioning process for defining objects and processes.

The process in Figure 3.5 starts with determining the problem domain-containing *things* that could form part of the system. An environmental object and non-transformable process loops back to the definition of a *thing*. If the *thing* is an object and systemic, it can be transformed and therefore forms part of the primary function. Similarly, a process that transforms an object also forms part of the primary function. After this, the primary function loops back to supporting the problem domain. The process in Figure 3.5 represents a unique requirements extraction view for the ISARC using OPM. Figure 3.5 applies the questions that define form in Crawley et al. (2016)(pp.68), function in Crawley et al. (2016)(pp.98), and systems architecture in Crawley et al. (2016)(pp.126). For brevity, the process appears in the enumeration below.

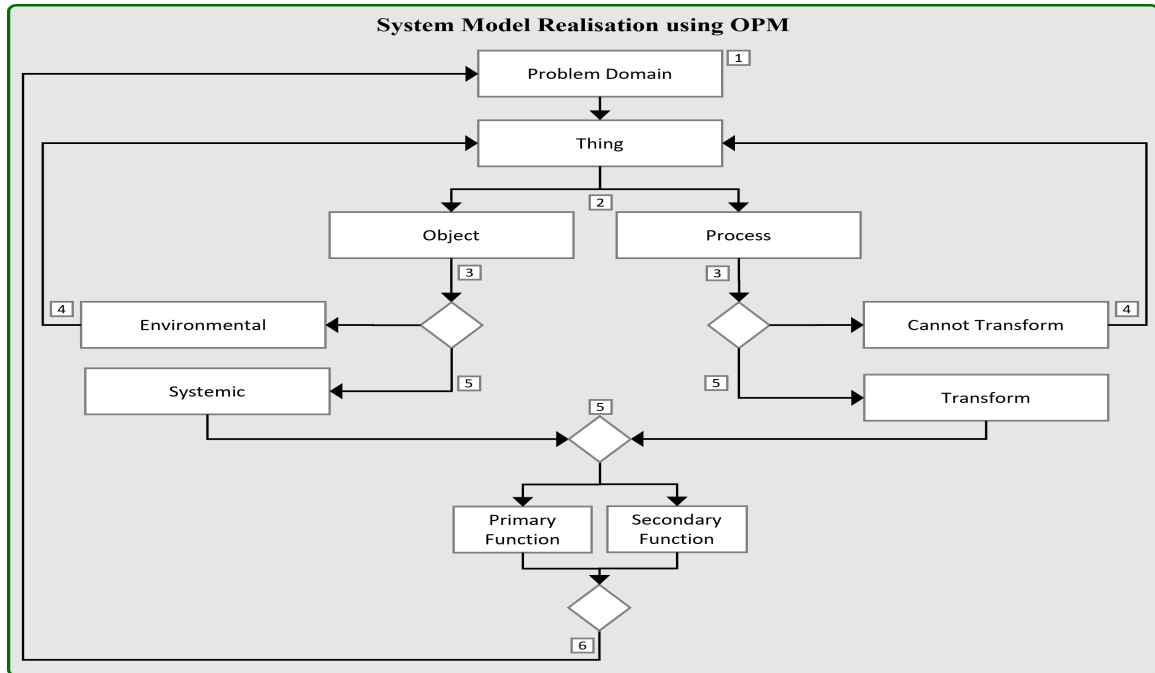


Figure 3.5: IoT-Ce System Model Realisation Flow. Adapted from Crawley et al. (2016) and ISO/TC 184/SC 5 (E) (2014)

1. Identify Problem Domain.
2. Is Thing Object (Enabler or Agent) or Process?
3. If Object, then is Systemic or Environmental? OR If Process, then Can Transform or Cannot Transform?
4. If Object is Environmental AND Process Cannot Transform, Identify Next Thing .
5. If Object is Systemic AND Process Can Transform Assign to Primary OR Secondary Function.
6. Return to Problem Domain.

Within Figure 3.2, the “OPM Thing” resides at the root node of the system’s hierarchy in OPM. In the OPM hierarchy, Things contain a state or are prone to execute a state change. The structure adaptation in Figure 3.4 references Crawley et al. (2016)(pp.101), where a specific state change is shown in the operand exhibiting state changes using process inputs and outputs. Thus, a reconciliation between the standard OPM and the critical questions in Crawley et al. (2016) - namely Form (pp.68), Function (pp.98), and Systems Architecture (pp.126), produces a set of incremental process steps. These steps culminate in step 6e of Crawley’s Systems Architecture questions in Crawley et al. (2016)(pp.126). This step yields

a primary or secondary function. The output of the cyclical activities in Figure 3.5 enables primary function identification and a recursive view of subsequent function identification by starting with the problem domain as the initial process step.

3.3 ISARC Vertical Theme Depth Convergence

OPM for the IoT-Ce

The focus to this point in Chapter 3 has been to set out a deeper position on the use of OPM ISO-19450. As supported in its views on the process of abstraction, Chapter 3 also argues that the nature of IoT-Ce **Front-end System Set** instances are suitably heterogeneous. This heterogeneity provides a general abstracted viewpoint that is plausible within reach of a single domain hypothesis. Henzinger and Sifakis (2007) believes that heterogeneity has two fundamental subsystem compositions; *varying interaction semantics* and *system abstraction*. Mordecai and Dori (2017a) handles this through the modelling of systems using a process called *zooming* and *unfolding* which are native in Object Process Diagrams (OPDs), ISO/TC 184/SC 5 (E) (2014)(pp.84). In his view on *complexity management* Dori (2016a)(Ch. 21) indicates that four key aspects of modelling aid in reducing complexity inherent in systems. These are; zooming-in and out, folding and unfolding, state suppression and expression, and view creation. Zooming relates to **process** refinement and **object** arrangement, whereas unfolding relates to structural object decomposition. The remainder of this section deals more specifically with the extraction of requirements for the IoT-Ce using the top-down approach indicated in Figure 3.5.

In a previous section on core IoT themes (section 2.2.1.1), a series of functions outline possibilities for the IoT, which innovatively grow out of pure IoT definitions. For brevity, these definitions restate IoT-Ce themes and immediately extract noteworthy **objects** and **processes** directly from these IoT definitions.

1. The definition of IoT as seen in a strategic research agenda by Vermesan et al. (2009) is: “an integrated part of future internet and could be defined as a dynamic global network infrastructure with **self configuring** capabilities based on standard and interoperable **communication** protocols where **physical and virtual “things” have identities, physical attributes, and virtual personalities** and use intelligent interfaces, and are seamlessly integrated into the information network.”
2. The definition of IoT, as seen in a strategic research agenda by a Finnish group, Ailisto et al. (2011), is: “a global network and service infrastructure of variable density and **connectivity** with **self-configuring** capabilities based on standard and interoperable

protocols and formats. IoT consists of **heterogeneous things that have identities, physical and virtual** attributes, and are seamlessly and securely integrated into the Internet.”

3. The definition of IoT as seen in a research overview by Miorandi et al. (2012), is that: “(IoT) is broadly used to refer to both: (i) the resulting global network interconnecting **smart objects** by means of extended Internet technologies, (ii) the set of supporting technologies necessary to realize such a vision (including, e.g. RFIDs, sensor/actuators, machine-to-machine communication devices, etc.) and (iii) the **ensemble of applications and services** leveraging such technologies to open new business and market opportunities.”
4. The definition of IoT as seen in an IEEE ecosystem study, IEEE (2015), that focused on the IoT market, technology and standards is that: “IoT refers to any **systems of interconnected people, physical objects**, and IT platforms, as well as any technology to better build, operate, and manage the physical world via **pervasive data collection, smart networking**, predictive analytics, and deep optimization.” Additionally, the definition includes that “for the most part, IoT devices will be **self-configuring** and adaptive to reduce the need for human interaction.”

Table 3.4 presents the extracted requirements baseline using a direct literature-to-OPM definition concept. The OPM Equivalent Candidate field in Table 3.4 holds proposed candidates that form the center of each form, function and architecture loop input and output noted in Figure 3.5. The OPM Equivalent Candidate field holds identifiers **O**: for object, **P**: for process, and where applicable, **S**: for state. In producing Table 3.4, a series of verifications against

Table 3.4: IoT Requirements Extraction Parameters

Source	Parameter	OPM Equivalent Candidate
A1.1	Self-configuring	P :Self-configuring
A1.2	Communication	P :Communicating
A1.3	Identifiable Things	P :Identifying, O :Thing
A1.4	Physical Attribute	P :Attribute Handling, O :Physical Attribute
A1.5	Virtual Personality	P :Personality Handling, O :Virtual Personality
A2.1	Connectivity	P :Connecting or Connection Handling
A2.2	Self-configuring	P :Self-configuring
A2.3	Heterogeneous Things	P :Heterogeneous Thing Handling, O :Heterogeneous Thing
A2.4	Identifiable Things	P :Identifying, O :Identifiable Thing
A2.5	Physical Attribute	P :Attribute Handling, O :Physical Attribute
A2.6	Virtual Attribute	P :Attribute Handling, O :Virtual Attribute
A3.1	Smart Objects	P :Smart Object Handling, O :Smart Object
A3.2	Applications & Services	O :IoT Application, O :IoT Service
A4.1	Interconnected	P :IoT Interconnecting, S :Interconnected
A4.2	People & Physical Objects	O :IoT User Group, O :Physical Object
A4.3	Pervasive Data Collection	P :Pervasive Data Collecting
A4.4	Smart Networking	P :Smart Networking
A4.5	Self-configuring	P :Self-configuring

OPM is possible using standard OPM definitions, ISO/TC 184/SC 5 (E) (2014)(section 3.39, 3.68 and 3.58). These are;

- Do the elements marked with an **O:** represent a thing that exists or might exist physically or informatically?
- Do the elements marked with a **P:** represent a transformation of one or more objects in the system?
- Do the elements marked with an **S:** represent a possible situation or position of an object?

For the first bullet, it is a resounding, Yes. One could argue that the entire IoT definition scope in list 1 to 4 can further extract into **O:**, **P:**, and **S:** elements. Considering the extent of possible proofs for the IoT-Ce, a limited parameter set yields a bound requirement set. Additionally, as noted in the earlier views on completeness and clarity, the models presented in this thesis generalise a method that handles an expansive body of knowledge for IoT.

Considering **P:** elements, the standard suggests the use of gerund verbs to describe the handling of an object, ISO/TC 184/SC 5 (E) (2014)(B.6.3). In his course work on OPM, Dori presents the use of the verb [handling](#) as a suitable construct to clarify process execution, Dori (2019a) and Dori (2019b). In the example of A2.5 and A2.6 in Table 3.4, the change of the parameter [Physical Attribute](#) to a process termed [Physical Attributing](#) would not represent a credible process. The notion of attributing versus the handling of the attribute would be the difference between identifying a thing based on its attributes instead of changing the state of one or more of those attributes. Hence, A1.4 in Table 3.4 in Table appears as [Attribute Handling](#). One could argue that the definition in A1.4 seeks merely to describe and not to change the attribute. This argument would be correct in terms of the virtual attribute such as changing an [Front-end System Set](#) internet protocol (IP) address or Media Access Control (MAC) address. Although, the latter is typically part of the underlying physical hardware.

Finally, concerning the use of the **S:** element, this element represents a single parameter termed [Interconnected](#) in A4.1. The primary question relates to the use of the parameter in the definition in the IoT definition item list 4; What is [Interconnected](#)? The definitions reads; “IoT refers to any systems of interconnected people, physical objects”, IEEE (2015). This definition indicates that the state of being [Interconnected](#) can be either not-interconnected

or **Interconnected**. A reference to not-interconnected means dispersed without a connection. Thus, the state is either that an interconnection exists or it does not. This state of a **declining interconnection** is a typical problem occurrence that identifies the need for a system. The system's purpose would ideally change the lower interconnection level to a higher interconnection level. A practical example in Sub Saharan Africa is the low levels of internet penetration, which could ideally be changed to higher levels of internet penetration using mobile technologies, with all other things being constant or environmental.

IoT Edge and Gateway Mobile Devices in OPM

For the IoT **Front-end System Set** requirements, the inclusion of the three primary focus domains mentioned in Chapter 1, Figure 1.1 adapted from Chuah (2014), is the primary utility of an IoT-Ce device. These **Front-end System Set** goals are to; generate data, acquire data, transmit data, and analyse data. Additionally, the conceptualisation of the edge architecture components from the EPC Global Architecture by Uckelmann et al. (2011) can also apply against the definition-to-OPM extraction. The notion of harder system boundaries for edge components is noted as Front-end, Near-end, and Far-end, Yu et al. (2017). Table 3.5 applies the same logic to extract the requirements using the definition-to-OPM structure employed in Table 3.4. The use of **P:** for a process, **O:** for an object, and where applicable, **S:** applies all the same definitions previously noted. The extraction process uses

Table 3.5: IoT-Ce Extracted Domain Parameters

Source	Parameter	OPM Equivalent Candidate
B1.1	Acquisition	P: IoT Data Generating and Acquiring
B1.2	Transmission	P: IoT Data Transmitting
B1.3	Analytics	P: IoT Data Analysing
B2.1	Filter and Collect	P: IoT Data Filtering, P: IoT Data Collecting
B2.2	Reader and Scanner	P: IoT Data Reading, P: IoT Data Scanning
B2.3	Sensor	P: IoT Data Sensing
B2.4	Actuator	P: IoT Control Actuating
B3.1	Front End	O: Front-end System Set
B3.2	Near End	O: Near-end System Set

the gerund guidance and provides initial questions on the aggregation of the requirements in Table 3.4. For example, in order for Table 3.5 B1.2 **Transmission** to deliver the requisite systems value, Table 3.4 A1.3 **Identifiable Things** might yield a dependency. Thus, a pre-existence argument emerges as - B1.2 **Transmission** requires A1.3 **Identifiable Things**. Although novel in this simple example, the ability to transmit data is generally underpinned by an identifiable device such that source and destination nodes in an IP-based system are known. In the purposeful absence of citing technical Transmission Control Protocol (TCP) or Open Systems Interconnection (OSI) protocols, the emergence of system value is directly pinned on using an OPM-based systems architecture viewpoint.

Additionally, if the question relates to non-IP-based communication, an onboard system register is needed for transmission between components within hardware-based lower-level CPU communication. The discussion on which part is the component, subsystem, and system in the hierarchy becomes a combination of questions on the pre-existence requirements for system value. However, a greater interest exists in hardware/software co-design and not pure hardware design. The content of Table 3.5 B1.1 to B1.3 also argues for the consideration of parameters such as Table 3.4(A2.2 Self-configuring). Self-Configuration is perhaps not a core requirement for B1.1, B1.2, and B1.3. Thus the A2.2 can be argued as either systemic or environmental. B3.1 and B3.2 uses a native **Object Group** and **Object Set** naming notation as seen in ISO/TC 184/SC 5 (E) (2014)(pp.117) - **Groups** represent human actors whereas **Sets** represent non-human objects. The parameters that form the next decomposition layer is the architecture is presented in the IoT capability overview in Figure 2.1. The parameter classes exist with three boundaries in Eclipse (2016). Two are the primary areas of concern; **Constrained Devices** and **Gateways**, thus in ISARC terms **Front-end System Set** instances and **Near-end System Set** instances. Table 3.6 produces the OPM equivalent of the Eclipse (2016) components.

Table 3.6: IoT-Ce Front-end System Set Extracted Parameters

Source	Parameter	OPM Equivalent Candidate
C1.1	Communication	P:Communicating
C1.2	Hardware Abstraction	P:Hardware Abstraction Handling
C1.3	Operating System	O:Operating System
C1.4	Remote Management	P:Remote Managing
C2.1	Data Management and Messaging	P:Data Managing, P:Messaging
C2.2	Communication and Connectivity	P:Communicating, P:Connecting
C2.3	Application Runtime	P:Application Runtime Handling
C2.4	Operating System	O:Operating System
C2.5	Remote Management	P:Remote Managing

Finally, IoT-Ce **Front-end System Set** requirements for device abstraction are present in the CPS body of work. The design space indicates that hardware/software co-design benefits from mapping actual system requirements to the allowable system requirements. Boundary conditions are essential stipulations in systems architecture as it creates the whole product system that should be modelled to attain a well-bound systems architecture. Interfaces exist at system boundaries to delineate form and interacting functions. Interfaces identify as component connectors, Kopetz (2011). Supporting this view, the model standards contained in ISO/TC 184/SC 5 (E) (2014), and implemented by Mordecai and Dori (2017a), indicate system boundaries as the definition across which subsystems or components interface. More specifically, the definition of a boundary is the line between systemic and environmental aspects of the system ISO/TC 184/SC 5 (E) (2014). Essentially, it “separates things that

are under the control of the system, its user or designer, from things that interact with the system but cannot be controlled by it“, Dori (2019b).

Table 3.7: IoT-Ce Front-end System Set Process/State Extracted Requirement Parameters

Source	Parameter	OPM Equivalent Candidate
D1.1	Notification	P:Notifying
D1.2	Request	P:Requesting

In **Front-end System Set** architecture, it is previously shown that interfacing requirements use a standard layered Android architecture in Figure 2.12. The ISARC focuses on the natural, accessible states that the Android operating system produces to reduce any inconsistencies in systemic or environmental arguments for **Front-end System Set** instance. As indicated in the Table 3.6, **C1.3 Operating System** and **C2.4 Operating System** generally appear across a single boundary, but due to the fit of mobile devices as gateways, the operating system option between IoT edge device and IoT gateway can consolidate within a single boundary - a **Front-end System Set**. This consolidation is suitable as noted in the earlier view on device abstraction by Latronico et al. (2015) and Lohstroh and Lee (2015). Synchronous and asynchronous implementation technologies using NodeJS on Android executes on both mobile client or server subsystems. Linux (2018) outline two primary classes of aggregated state handlers; *Notifications* and *Requests*. In definition-to-OPM form these are; **Notification**(O) and **Request**(O), and **Notifying**(P) and **Requesting**(P). Table 3.7 produces a condensed system requirement set for process and state awareness using Android-based NodeJS constructs.

Runtime Configuration in OPM

The CPS research agenda (ARTEMIS) has a specific role in delivering design by composition as opposed to decomposition as indicated in Schoitsch and Niehaus (2017). CPS challenges are similar to those in IoT. The challenges in CPS modelling arise due to the heterogeneous nature, concurrency of different physical processes, and real-time requirements, Khaitan and McCalley (2015). The composition and arrangement of variables in a system to meet these system goals represent the configuration, as previously noted in section 2.2.3.

The modelling of the runtime possibilities of the **Front-end System Set** interaction process uses a known process baseline seen in general process transition. The detail of the process transitions extracted from Reichert and Weber (2012)(pp.34) closes the abstraction of flows of processes in a **Front-end System Set** by binding the possible process transitions into a finite state space. Table 3.8 represents this set of finite process transitions and state changes.

Table 3.8: IoT-Ce State Transition Extracted Requirement Parameters

Source	Parameter	OPM Equivalent Candidate
E1.1	Created	P:Creating
E1.2	Running	P:Running
E1.3	Suspended	P:Suspending
E1.4	Completed	P:Completing
E1.5	Activated	P:Activating
E1.6	Terminated	P:Terminating

Each of the parameters in Table 3.4, 3.5, 3.6, 3.7, and 3.8 support the assignment of a question relating to the role that a single or string of parameters plays in the ISARC. For example, to produce the system value of A2.1, B1.2 is a necessary process, of which C1.x parameters are required. Once identified, both D1.x parameters might be required to be in a specific E1.x state to produce the system value. In mapping the objects, processes, and states, it is possible to decompose the system value question into a running process by using consistency in the extraction of parameters using a hierarchy, boundary, and OPM.

Finally, additional parameters relating to the argument of configuration before reconfiguration is handled by applying the definition-to-OPM extraction approach against the IBM MAPE-K. Arguments for adaptive systems through the use of IBM’s MAPE-K framework is seen in Cheng (2008), Swanson et al. (2014), Ashraf et al. (2016), and several references in adaptable systems literature in Villegas et al. (2017). These references produce views on adaptive systems dealing with failure conditions—the capabilities to support configuration variation functions at both runtime and design time. The runtime behaviour of the IoT-Ce is the operational behaviour in that the complex system acts dynamically. This complex systems view is supported by Crawley et al. (2016). Furthermore, the system interacts in a sequenced manner internally, as well as with supporting external architectures. These behaviours require considerations for operators, event sequences, and the operational cost. Hence, the runtime configuration of a system is defined as the dynamic arrangement of the system while the IoT-Ce **Front-end System Set** is in operation. Extraction of parameters from the MAPE-K, IBM (2005) delivers the candidates parameters in Table 3.9. It has already been shown that the Cheng (2008) body of work delivers a specific change on the types of knowledge required that is noteworthy in the argument relating to configuration before reconfiguration. As indicated in Cheng, knowledge of the solution topology, handling of change policy, and a series of problem pre-determinants aid successfully in the overall adaptivity of the system.

The inherent process order of the MAPE-K relates the MAPE-K as a function that transforms an object state. As indicated in the earlier view on the definition of systems

Table 3.9: IoT-Ce Control Extracted Requirement Parameters

Source	Parameter	OPM Equivalent Candidate
F1.1	Monitor	P:Monitoring
F1.2	Analyse	P:Analysing
F1.3	Plan	P:Planning
F1.4	Execute	P:Executing
F1.5	Knowledge	P:Knowledge Handling, O:Solution Topology Knowledge, O:Policy Knowledge, O:Problem Determination Knowledge

architecture in Figure 3.4, the [Monitoring](#) process is an action that changes the state of an operand [Knowledge unknown](#) to [Knowledge known](#). F1.1 [Monitoring](#) interacts with [Knowledge](#) to interact with F1.2 [Analysing](#).

The section dealt in depth with a requirement extraction process for the ISARC. The assumption that the extraction approach can handle both simple and complex interactions is correct. The basis for ISARC requirements extraction is a normative approach to consuming a large body of work and filtering this into a natural hierarchy with associated boundaries. Tables 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9, produces a filtered parameter set that bind the IoT-Ce into objects, processes and states. The consequence of including only six parameter output tables creates a suitable boundary for model output; however, it gives rise to possible requirement gaps and hidden concepts. Under the use of UML or SysML the handling of requirements would have to be traced from prior model types. OPM applies *Mayer's Cognitive Assumptions* for dealing with continuous learning about the system under specification, Dori (2011). For brevity on applying these principles in the ISARC requirements extraction section, the order of application is; (1) active learning while specifying the system, (2) limiting channel content and (3) dual-channel processing. Dual-channel processing becomes applicable in Chapter 4 during the modelling of the extracted requirement set where both OPD and OPL generate at the time of model specification. A consolidation of the tables in the section is located in Table B.1.

3.4 System Description

A system description is the description of a problem that necessitates finding a system that solves that problem, Dori (2019a). A problem occurrence model is a two-level OPM model, separate from the solution model, and defines a systemic process in the System Diagram (SD) of the problem occurrence model. The appropriate system value statement presents a decomposable statement from the highest hierarchy node in the OPM SD hierarchy. A suitably abstract example for the top-level node is [IoT Edge System Value Delivering](#). This statement indicates that some value is delivered from the system. The alternative would

be to indicate **IoT Edge System Value Abandoning**, which is consequently the opposite of trying to deliver value. In the preloaded example system cases in OPCAT 4.2, top level system value is seen;

1. in an ABS example where **ABS Braking** changes **Velocity** from **high** to **zero**,
2. in an Alzheimers example where **Dementia Diagnosing & Managing** affects **Patient**,
3. in an ATM example where **Transaction Executing** yields either **Transaction** or **Denial Notice**, and
4. in an Order Management System example where **Product Handling** yields **No Product Message** and **Order**.

Given the primary thesis objective that suggests that IoT definitions are overtly conceptual and avoids core system value, the representation of parameters from the IoT definitions mentioned in Tables 3.4 and 3.5 yields a suitable notional value. For the system description, Tables 3.6, 3.7, and 3.8 provide additional and supporting value-providing OPM *things*. In obtaining a larger simulated selection criteria, the second contribution aims to provide the Python codeblocks that deliver continuous decision model simulation.

As a primary system value, the central thesis contribution indicates that the IoT-Ce should provide some form of value and hence, **IoT Edge System Value Delivering**.

3.4.1 System Form and Function

The extraction of processes from the core IoT definitions seen in the requirements extraction section presents processes such as **Self-configuring**, **Communicating**, **Identifying**, **Attribute Handling**, **Connecting**, **Interconnecting**, **Transmitting**, **Analysing**, **Sensing** and **Actuating**. Note the use of gerund verbs. Additionally, forms such as **Front-end System Set** and **Near-end System Set** are suggested as primary instruments for delivering systems value. Table 3.10 introduces a function identifier - **F:** - that directs a change of the operand state. The possible problem occurrence outlined in the IoT Form-Function overview is coupled into the existence of a device (**Front-end System Set**), when device identification occurs (**Identifying**), and when device communicating occurs (**Communicating**), in function cluster 1 (Ax). One could argue that **Front-end System Set Environment Sensing** and **Actuating** occurs before communication. However, this function would require the existence of an instrument such as **Front-end System Set** in the function **Front-end System Set Space Occupying**, nonetheless. Here a simple process order drives the primary system process of **Front-end System Set**

Table 3.10: Consolidated Form-Function Interaction

Source	Form-Function Candidate
A1.1	F: Self-configuring changes Front-end System Set from non-operational to operational
A1.2	F: Communicating changes Front-end System Set from inactive to active
A1.3	F: Identifying changes Thing from unidentified to identified
A2.5	F: Attribute Handling yields a Managed Physical Object
A2.6	F: Attribute Handling yields a Managed Virtual Object
A2.1	F: Connecting changes Device from not-connected to connected
A4.1	F: IoT Interconnecting changes IoT-Ce from dispersed to interconnected
B1.2	F: IoT Data Transmitting changes Data Object from non-transmitting to transmitting
B1.3	F: IoT Data Analysing changes Data Object from non-analysed to analysed
B2.3	F: IoT Data Sensing changes Data Object from non-sensed to sensed
B2.4	F: IoT Control Actuating changes System Object from non-controlled to controlled
B3.1	O: IoT Edge System Value Delivering requires Front-end System Set
B3.2	O: IoT Edge System Value Delivering requires Near-end System Set
C1.4	F: Remote Managing changes ED from unmanaged to managed
C2.1	F: Data Managing and Messaging changes data from unmanaged to managed
C2.5	F: Remote Managing changes GW from unmanaged to managed

Identifying, which produces a transformed operand of **Front-end System Set Identity changing from unidentified to identified**. One could further extend the argument to produce a problem occurrence aligned to the IoT-Ce to produce a non-integrated system against which IoT-Ce will need to produce an integrated system value set. The purpose could therefore be to increase automation (**Self-configuring**) across one or many of these functions.

With the primary objects represented as a set of Physical Devices (Smart Thing, Smartphone, or ED/GW), one could model the system's purpose as self-configuring given a primary value driver such as **Seamless Connecting**. As per the extensive standardisation efforts, an external environment impacts the IoT-Ce as the IoT System consumes these standards to deliver services to users. An incorrect IoT Protocol Set could result in a problem occurrence of *Increasing Device Connection Time Handling*, for which **Seamless Connecting** is necessary.

Considering a final example that a definition-to-OPM extraction serves to filter general viewpoints in IoT-Ce, a representation of a form and function extract outlines the domain classes from Sridhar (2013). The traditional functional system domain classes exist in a layered architecture. This functional domain architecture extends to the aggregated architecture options for specifying IoT Device, IoT Gateway, and IoT Cloud subsystems indicated in Figure 2.1. Thus, the overall coverage of the architecture exists between a logical view of an IoT architecture and a conceptual view of the systems architecture. The table containing the mobile cloud form and function overview creates a similar view to the IoT form/function overview in Table 3.10, but instead uses the object (Candidate Form) as the primary entry to the ISARC. Cluster 1 to 3 in the mobile cloud form and function table represents edge and connectivity systems. It excludes cloud components such as infrastructure, organisation,

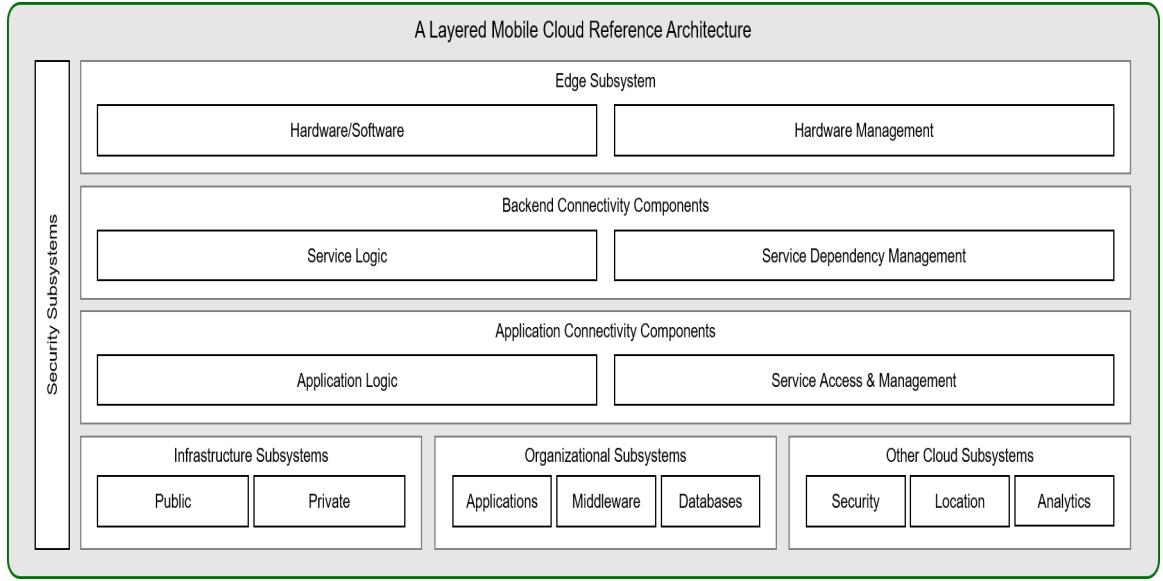


Figure 3.6: Mobile IoT Cloud System Domain adapted from Sridhar (2013)

Table 3.11: A Mobile Cloud IoT Extracted Requirement Parameters

Candidate Form	Function
Cluster 1	
Hardware	Hardware Space Occupying
Software	Software Logical Space Occupying
Hardware Management	Hardware Managing
Cluster 2	
Service Logic	Service Logic Executing
Service Dependency Management	Service Dependency Management Handling
Cluster 3	
Application Logic	Application Logic Executing
Service Access	Service Access Handling
Service Management	Service Managing

and other cloud systems as these systems are part of the core architecture and not the edge. Additionally, to validate a modelled set of functions, it describes behavioural dynamics that can appear at SD1 or lower-level OPD.

3.4.2 System Interfaces & States

An Interface Theory for CPS by Lohstroh and Lee (2015), implements IoT systems concurrency and timed-behaviour. The monitoring of state within a parallel flow of events or an event-dependency flow is necessary to execute a level of control to mediate the flow of events to maintain flow integrity. It is a common control reference point comparison, as seen in general control theory.

To support the delivery of components in IoT edge systems, several IoT programming lan-

guages have found their way into IoT prototyping and real-world implementations. One such language is a suitable candidate as it serves both CPS research and single language client-server architectures. This javascript framework, known as NodeJS, is built on Google's V8 runtime engine, Cantelon et al. (2014). NodeJS has had considerable reach in both academic and industry circles. The most notable being the implementation of the actor based accessor architecture for CPS integration, Brooks and Lee (2017). Synchronous and asynchronous implementations using NodeJS can run on both client or server subsystems. In the case of IoT-Ce **Front-end System Set** architecture, NodeJS operates on Android. Thus, as a technology implementation, it applies as a key component in delivering deployable units for accessing edge components. Figure 3.7 outlines a stepped process used in NodeJS to achieve

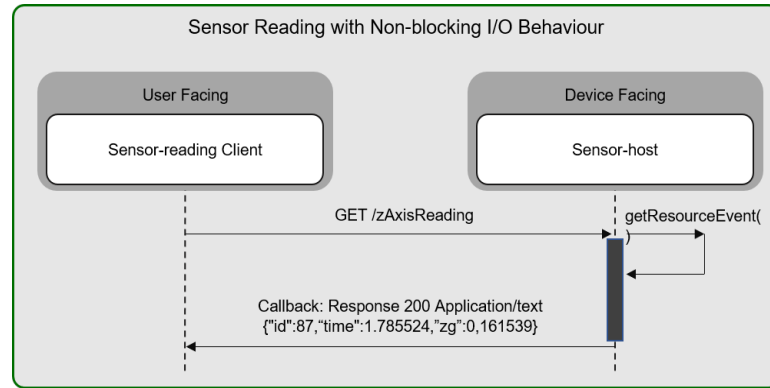


Figure 3.7: Non-blocking Application IO

non-blocking behaviours. A similar flow is used in CoAP, mentioned in Kostic (2017). The flow shortens as;

- a request for a resource is made to the Javascript server components,
- an event is triggered,
- another IO operation completes, and
- the resource returns via a callback to the requesting component.

NodeJS has a large community of open source developers, supporting more than two hundred thousand libraries. Device libraries such as OS and OSENV rank consistently in the top 1000 Node Package Manager (NPM) dependency downloads. These libraries produce unique operating system access APIs that abstract the core operating system technology, Linux (2018). For example, the NodeJS OS package libraries;

- display device interfaces using the following command: `require('os').networkInterfaces()`,

- display the uptime of a system using `os.uptime()`,
- display the total system memory using `os.totalmem()`, and
- display scheduling priority for the process specified by process identifier (pid) using `os.getPriority([pid])`.

Access to this library allows an application, such as an ED/GW, to access a device over standard TCP/IP protocol interactions. Both NodeJS and NPM libraries can execute on IOS, Android, and Windows. Thus, the integration possibility of this language as a primary or secondary implementation language for IoT systems is considerably high. Owing to the native support for web client-server architectures, it also extends the reach of deployment possibilities across Windows and Android devices.



Figure 3.8: Samsung Galaxy running NodeJS in Termux

Figure 3.8 shows a command-line execution of the NodeJS OS library on an Android Device.

A total of 32 combined OS states are included in the Node.js v11.2.0 Documentation, Linux (2018). Core error states and fault notification can be listed by **Notification** and **Request** types, as noted in the earlier parameter Table 3.7. Thus, the 32 OS state constants provides a further set of attributes for expression of **Front-end System Set** behaviour. Therefore, the basis of the allowable Android error and fault states provides a set of finite states, accessible device data, and system interactions against which ISARC parameters from Tables 3.8 and 3.9 can derive a clean systemic interaction parameter set.

Boundaries and Conditions

Boundaries are essential to the definition of a system, Crawley et al. (2016). Boundaries characterise by;

1. an operand passing through the boundary,
2. the process at the boundary is usually common/shared, and
3. identical or compatible interfaces on either side of the boundary.

Figure 3.9 indicates a conceptual overview of the boundary adapted from Figure 3.4 in Crawley et al. (2016), showing a possible decoupling of the aggregated architecture shown in Figure 2.1. Figure 3.9 represents the **Notification Connecting** function as a shared func-

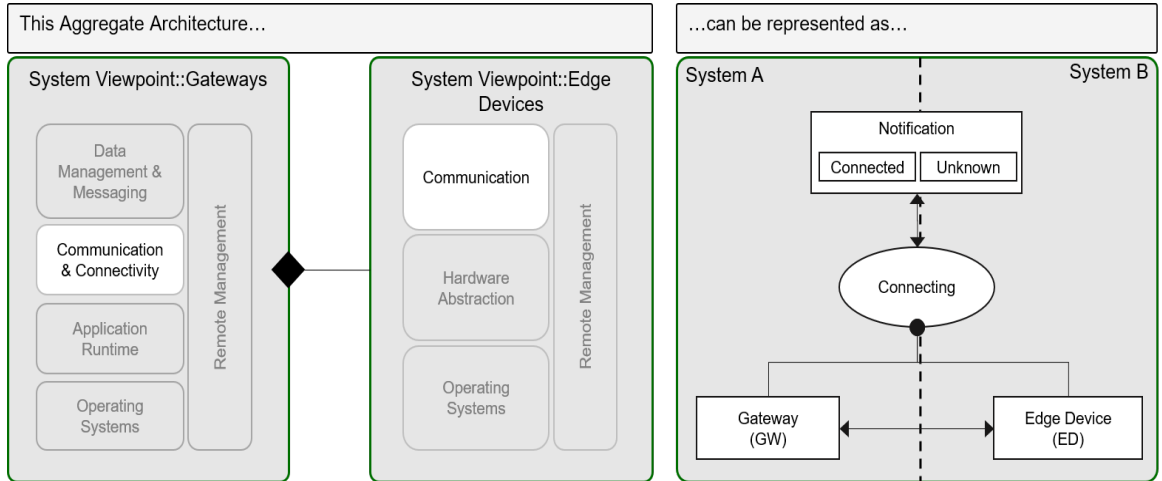


Figure 3.9: System Boundary Mapping adapted from Crawley et al. (2016)

tion. **Communication** and **Connectivity** are aligned with **Communication** in the aggregated architecture option. The **Connecting** process at the boundary affects the **Notification** object. **Connecting** requires both **Gateway** and **Edge Devices** that have compatible interfaces on either side of the boundary. This boundary refers to a physical boundary in the case of a split **Gateway** and **Edge Devices**. The case of the IoT-Ce **Front-end System Set**, the

device houses both **Gateway** and **Edge Device** capability in a single physical mobile casing as exhibited in Figure 3.8.

Additional value for handling state comes predefined in OPM. Modelling of systems that adhere to bounded principles indicate that processes can preserve state of objects in a system. Processes are listed as those that maintain a steady state rather than transitioning between states, (Dori (2002)(pp.316)). Where needed, the processes that can maintain state appear as “**Supporting, Holding, Maintaining, Keeping, Staying, Waiting, Prolonging, Delaying, Occupying, Persisting, Including, Containing, Continuing, Enclosing, Fastening, Connecting, Postponing, Dragging, Storing, Owning, Restraining, Drawing, Attracting and Remaining**”.

3.4.3 Configuration Variation

In establishing the configuration space, a simple configuration problem would be assembling a thing or system out of an available set of components or subsystems. Petrie (2012) likens this problem to building a “boat out of a given set of lego blocks”. Further to this, Petri likens the configuration problem space as too large if too many components and combinations are apparent. For smaller problem spaces, Petri recommends the use of a configuration problem solver. A configuration problem solver is a computational item that models configuration items using constraints to satisfy a set of system requirements. Early automated configuration systems gave rise to a series of configuration requirements. Petrie (2012) summarises these as;

1. *limiting component functionality in order to limit the configuration space.* Where the reduction of component interactions yields a reduction of complexity, the use of OPM boundaries enables such limiting behaviour, ISO/TC 184/SC 5 (E) (2014)(section 6.1.5),
2. *semantic modelling to represent knowledge about the configuration space* as opposed to the rules that govern configurations. The use of OPM Thing and Link syntax and semantics enable this requirement, (ISO/TC 184/SC 5 (E) (2014)(sections 7 and 8),
3. *models to represent static knowledge about components.* The use of OPM Structural and Procedural constructs enables this requirement, ISO/TC 184/SC 5 (E) (2014)(sections 10 and 9), and
4. *component hierarchies and decomposition to separate components from control knowledge.* Several specific sections in ISO/TC 184/SC 5 (E) (2014) enable this requirement, specifically, Function versus Behaviour (sections 6.1.4), Kinds of fundamental structural relations (section 10.3.1), and Relationship cardinalities (section 11).

The configuration is an ordered collection of things that provide a set of constraints to a configuration problem. This position satisfies the problem requirements and constraints associated with the structural aspects of the form and function of the IoT-Ce. The segmentation of configuration problem-solving represents the structural aspects of control constraints. This segmentation is present in the characteristics of general and specialised configuration problem-solvers, Petrie (2012).

Thus, the view on configuration concurs that;

1. a configuration is the arrangement of all variables that require measurement, Smolin (2014), and
2. different types of (re)configuration exists, Jacobson (2004), and
3. that the composed arrangement yields a purpose and function, Dori et al. (2019), and
4. that some form of boundary exists around the system, it therefore indicates that,
5. *the arrangement of elements, or things, within the boundary that makes up the system is, in fact, a changeable configuration with options that produce the best value to the system beneficiary.* The IoT-Ce system value manifests through parameters such as those indicated in Tables 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9.

Additionally, given a set of pre-existence requirements, understanding the arrangement of components is essential to support ongoing change. The definition of the types of configuration supports this pre-existence requirement, as indicated in the section on configuration types in Jacobson (2004).

The systems view supports the overarching claim regarding pre-existence that objects can exist without processes in OPM, ISO/TC 184/SC 5 (E) (2014)(section 7 OPM Thing Syntax and Semantics). Thus the process to model self-adaptation requires the existence of some predefined object. Therefore, it follows that the **Front-end System Set** requires a **Configuration Set** as the configuration contains a hierarchical structure based on OPM symbolic structural requirements.

The arrangement of IoT-Ce system components previously argued for is proposed as an important factor in the change of IoT-Ce system behaviour, underpinned by a change in the state of the system and associated parts. The IoT-Ce realises as an ecosystem of interconnected parts. Due to its inherent variability, the modelling of an IoT-Ce system should be

considered an input to the primary indicators for the type of change that a running system can handle. More so, those aspects of the system that can change considering the minimal impact on the functions of the system. Therefore, the modelling of the system must provide an adaptable set of interfaces which integrates faster changing standards, with more robust higher-purpose standards. This form can be seen in section 3.4.2.

3.5 Architecture Decisions and Evaluation Criteria

The ISARC requirements are governed by a function to form mapping, which meets required systems architecture constructs. Thus, in order to maintain a well-structured system, it has to exhibit certain characteristics firstly;

1. **Representative** through the completeness of the model containing all necessary elements in order to suitably represent the Problem, Purpose, Primary Function, Enablers, and Environment.
2. **Flexible** using a modular, hierarchical set of entities that are structured to produce System–Subsystem–Component specifications,
3. **Bounded** to ensure non-cascading failure conditions, achieved utilising well-defined functional boundaries,
4. **Measurable** employing context-based output to obtain reliability measures.
5. **Repeatable** indicating that it can be validated effectively more than once.

DSMs for Architecture Decisions enable these properties through its compact structure and gearing towards qualitative and quantitative methods. The DSM structure supports the application of parameter-driven calculations that can simplify using OPM. Thus, the inclusion of possible parameter interactions from the existing body of work is guided firstly by fixed interactions before finding hidden links between decisions. Furthermore, Crawley et al. (2016), Eppinger and Browning (2012) and Washizaki and Tei (2019) position the importance of model integrity, visualisation and segmentation of architecture decisions, respectively. This approach permits decision structure evaluation criteria.

The definition-to-OPM extraction approach proposes applying an extraction method to filter system parameters from the IoT body of work. The assignment of these parameters into a decision model enables a fixed design, configuration, state, and decision spaces for a period. In refining the model, increases in the understanding of the system domain change

over time. Today, primary IoT literature exhibits an agreement that EDs are lossy and require GW components to interact. The ISARC blends ED and GW technology boundaries into a single **Front-end System Set**. In the not-so-distant future, mobile device sensor hubs become the basis for all new IoT-Ce use cases. As the study aims to create a cross-sectional time-bound view of IoT-Ce objects and processes, the selected literature in chapters 1 and 2 binds all the necessary parameters for a fixed period. The definition-to-OPM extraction approach extracts a series of objects, processes, states, and functions that proves a preliminary view of the applicability of the OPM for IoT-Ce system architecture specification. IoT-Ce evaluation criteria that apply the decision method in Kazman et al. (1998) integrates the parameter-based architecture analysis method by producing OPM specific things such as;

1. present parameter configuration space for primary IoT-Ce domain scenarios,
2. link parameter space to extract of known subsystem interactions, indicated in figure 2.1 and 2.11,
3. extract primary function parameter coupling with error limits,
4. extract secondary function parameter coupling with error limits,
5. include concrete control parameters,
6. model sensitivities & architecture tradeoffs, and
7. battle tests the model output against a historical data set!

In summary, the output of the hierarchical system model is used to generate a data set that evaluates against a historical data set to indicate the model's applicability for generating data for validation of hierarchical IoT-Ce systems.

3.6 Chapter Summary

Chapter 3 produces the core integrated system requirements for the ISARC. Preceding chapters positioned the vast body of work for concepts such as Systems Architecture, IoT, IoT-Ce, CPS and Embedded Systems Design, and Runtime Configuration. However, Chapter 3 covers the first integrated view of these vertical concepts. Several domain concepts, such as model compliance, model requirements filtering, and system descriptions, are explored. The definition-to-OPM requirements extract approach provides an output that produces both simplicities in the candidate parameter extraction process and creates singular parameter outputs. These outputs enable comparative qualitative analysis possibilities between somewhat fragmented parts of the IoT-Ce. The extract process enables qualitative comparison

for singular object, process, and state arguments. Backed by core systems viewpoints, the OPM enables questions on the hierarchical positioning of parameters. This placement serves as a vital modelling characteristic in a heterogeneous ecosystem, such as the IoT.

Chapter 3 also produces the realisation of a single model, guided by Dori's key system concepts; *problem occurrence*, *system purpose*, *primary function*, *human and system enablement*, and *environmental factors* in a system. The contribution towards the research goal has recommended the OPM as a means to simplify model complexity, enable greater usability, support modelling of reusable objects, and the goal of qualitative to quantitative structure for use in subsequent chapters. In terms of modelling of the IoT-Ce, it suggests that;

1. **for simplification** in resolving human cognitive constraints and enabling of mechanisms to cope with complexity using configuration. The OPM ISO 19450 produces single model constructs that enable active learning while the system is under specification.
2. **for simplification**, the use of co-design between software and hardware to reduce inconsistencies is also a known juncture between IoT and embedded systems. The OPM ISO-19450 models conceptual structures using primarily object, process, and state constructs. The OPM further enables hierarchical positioning using refinement methods such as zooming and unfolding.
3. **for greater usability** by focusing on a population that contains usable components such as mobile devices and those used in discovery and closed-loop communication. The OPM models single or multi-boundary systems by stating systemic and environmental objects, processes, and state placement. Lower-level process execution requirements are drawn from a known state transition model.
4. **for reuse** through the use of existing standards to meet the needs of IoT use cases. For example, the use of resolvable addressing protocols in Domain Name Systems supports Object Naming Services for edge-device identification. Additionally, the reuse of internet protocol-based interaction standards such as web services communication using the Constrained Application Protocol (CoAP) and NodeJS that exposes Android state and process constants. Additionally, OPM enables dual-channel model-to-text outputs that are reusable in text mining activities.
5. **for management and optimisation** of smart object interaction across dimensions such as state and policy awareness. The OPMs models of the IoT-Ce enables system

testing of models using parameter limits and logical control flows such as exception links.

The key parameter outputs of Chapter 3 concur with standards related to Object-Process Method, as required in the OPM ISO-19450 standard. The chapter outlines the system as containing key concepts such as the IoT-Ce, runtime configuration, and architecture decisions that will suggest the delivery of a well-structured model. The system requirements outline the application of system modelling to a heterogeneous system, using OPD and OPL to structure the foundation requirements, which support subsequent chapters. The systems viewpoint argued in this chapter concurs with the course views by Professor Dov Dori (2019a) - the creator of OPM. They are;

1. the system structure is static, independent of time and represented using structural links,
2. the system behaviour is dynamic, dependent on time and uses procedural links, and
3. the combination of structure and behaviour enables its function, which describes the System Architecture.

Chapter 3 serves to build the requirements for modelling, prototyping, and validating a Systems Architecture for the IoT-Connected Edge. It concurs that the arrangement of elements, or things, within the boundary that makes up the system is, in fact, a changeable configuration with options that produce the best value to the system beneficiary.

Chapter 4

Modelling the ISARC: Systems, IoT-Ce and Configuration

The preceding chapters have displayed the broad scope and reach of IoT systems. Therefore, producing an accurate system value can vary depending on whom the stakeholder group achieves benefit from the system. Chapter 4 models the requirements stated in Chapter 3 into several OPM SDs (System Diagrams), labelled the **IoT-Ce Systems Architecture for Runtime Configuration** (ISARC). With many IoT architectures available in industry and academia, the modelling of the decision impacts of different configurations is necessary to enable greater choice of how to deploy these IoT systems into real-world scenarios. Decisions on the boundaries between edge and core IoT architecture derive through conceptual frameworks. Frameworks such as the Extended Electronic Product Code (EPC) Global Architecture, Chuah's perspectives on functional component placement, Yu et. al.'s views on component relationships for edge users, and the wave of intelligent mobile devices that increase utility for average smartphone users have proven as suitable coverage for IoT-Ce arguments. The extensive coverage in the IoT body of work has produced an openly conceptual view without the requisite system value to connected edge users. Additionally, frameworks that guide architectures for autonomic systems such as IBM's MAPE-K contain several predefined components with clear functional boundaries. However, aspects of systems architecture are missing as the MAPE-K provides the basis of system behaviour change, such as failure avoidance, instead of user-specific systems architecture goals.

Chapter 4 proves that conceptual models can guide true systems architecture viewpoints. It explores the dual-channel visualisation of OPM by expanding the IoT-Ce requirement parameters extracted in Chapter 3. OPM has seen considerable success in complex systems design. However, value is yet to be shown for its use in the IoT-Ce.

4.1 System Purpose & Primary Processes

The inspection of the IoT architectural frameworks, approaches, and guides in the preceding chapters show considerable optionality in the derived parameter sets. Some parameters directly impact system value, whereas others do not. This impact delineates a clear separation of concerns. Considering the definitions of systems, subsystems, and components, a set of parameters find their way to the top of a parameter list, indicating different importance for different stakeholders. For example, in the case of a user of a Front-end System Set, the value of increasing communication dependability might supersede increasing security. However, in the case of the manufacturer of a Front-end System Set, both increasing dependability and increasing security might supersede increasing battery efficiency. There is always a tradeoff, and thus, understanding the parameter bias between competing decisions offers reprieve on the cost of applying a parameter selection. Competing architectural options for the ISARC look at the core functional boundaries produced from IoT frameworks. In line with the guidance on system architecture specification in Crawley et al. (2016), an outline of the function-to-form mapping of the ISARC uses system objects, relationships, interfaces, and peripheral systems. These state specifically as;

- the expansion of the IoT-Ce requirements to question placement in the ISARC hierarchy,
- the main system purpose and primary system interactions,
- supporting objects and process transformations,
- system interfacing and interactions that support control model types, and
- runtime configuration variations.

The architecture of a system derives using composition and decomposition, relative placement, coverage, boundary and interfacing, and use, Crawley et al. (2016). In stating a series of system considerations, one can elect to start modelling at different levels within the system hierarchy. In OPM, and similarly, in the ISARC, the top-level system diagram outlines the primary transformation and problem statement that the system aims to fulfil. The use of OPM to state IoT system value is, of course, as challenging to pin down as the primary system beneficiaries of IoT itself. Therefore, the SD - the root node in the OPM hierarchy - stipulates the broader system value of IoT as providing some form of value, namely IoT Edge System Value Delivering. This value point is explicit to provide a starting point at the apex of the vast body of knowledge seen in earlier chapters. This approach is essential in placing a marker at a point in a possible value space. This point is vital in understanding

the impacts of lower-level objects, processes, and states in connected SD diagrams. Finally, this approach positions the appropriate system architecture questions that deliver a system within a reasonable impact on its environment. These questions derive as an adaptation of the systems realisation flow seen in Figure 3.5.

System coverage characteristics reference Crawley et al. (2016). In their view on a set of coverage questions, architectural classifiers distil from questions relating to form, function, and the system architecture. The proposed IoT-Ce system value outlines a series of considerations aligned to the systems architecture, form, and function questions. As the core thesis argument delivers a systems architecture supported by the form to function map, the list below reverses the chapter flow seen in Crawley et al. (2016) to provide the core systems architecture questions first before being attached to the form and function questions stipulated in Crawley et al. (2016).

Table 4.1: IoT-Ce Form Function Adaptation. Adapted from Crawley et al. (2016)(pp.68, 98, 126)

ID	Classifier	Requirement
3.1	System Architecture	Main Object Process Transformation
3.2	System Architecture	Additional Object Process Transformations “non idealities”
3.3	System Architecture	Supporting Object Process Transformation
3.4	System Architecture	System Boundaries and Interfaces
3.5	System Architecture	Process Execution and Priority
3.6	System Architecture	Parallel Process Execution and Priority
3.7	System Architecture	Timing and Constraints
1.1	Form	Abstracted System Representation
1.2	Form	Decomposable Form Elements
1.3	Form	Spatial or Connectivity Structures
1.4	Form	Main and Supporting Systems Composition
1.5	Form	System Boundaries and Interfaces
1.6	Form	Use Context
2.1	Function	Primary Externally-delivered Value
2.2	Function	First and Second level Internal Objects, Processes and States
2.3	Function	Functional Architecture through Internal Process Interactions
2.4	Function	Secondary Externally-delivered Value

4.1.1 System Diagram (SD): IoT-Ce System Purpose

Dual-channel representation in OPM contains Object Process Diagrams (OPD) and Object Process Language (OPL) - graphical and text representations, respectively, ISO/TC 184/SC 5 (E) (2014)(pp.4 and 5). The IoT-Ce SD is shown in Figure 4.1. This SD represents the top-level node against which the first modelled SD1s are attached. The SD essentially decomposes to greater detail on the IoT-Ce front-end and near-end devices in SD1 and subsequent SDs. The primary problem statement stipulates that **dependability** is **low** and changed to **high**, and that the delivery of value for IoT requires a set of system

components - a **Front-end** and **Near-end System Set**. The process of delivering value to an **IoT System User**, is shown through increasing the dependability of the system. This primary purpose could have been to increase security, or increase battery efficiency, increase usability or any other non functional requirement. The primary purpose could also be to increase **self-configuration**, which would be applicable if the problem occurrence stated as **increasing human intervention yields an increase in connectivity failures**. Therefore **device self-configuring** is needed.

Conversely, a final state of; decreasing heterogeneity in a system that exhibits increasing heterogeneity would yield an SD that contains a different problem, purpose, primary process, enabler set, and environment representation. One can challenge the final state of heterogeneity by indicating that a stable set of heterogeneous standards, methods, and technologies could emerge. However, the primary challenges in the areas of IoT and CPS show that heterogeneity is increasing. Hence, component abstraction has become a requirement in the broader research agenda for IoT and CPS.

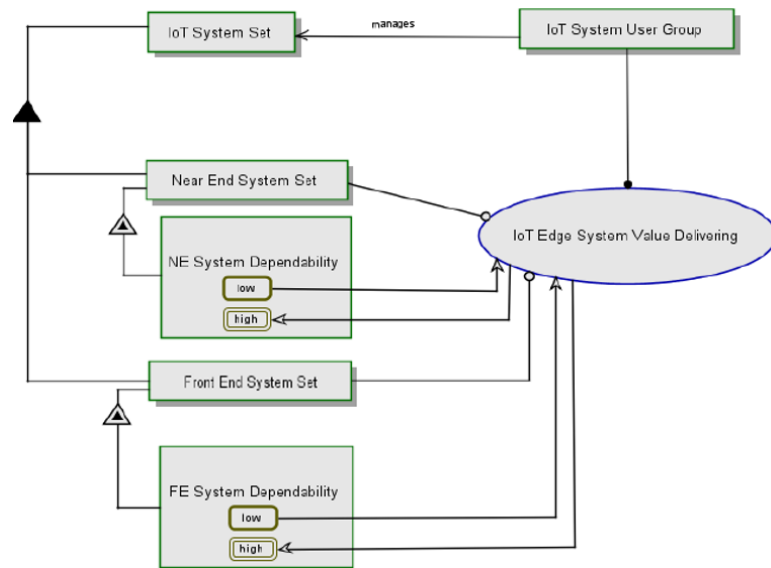


Figure 4.1: Sytem Purpose:SD IoT System Value

Listing 4.1: IoT System Purpose OPL

```

1 IoT System User Group is physical.
2 IoT System User Group manages IoT System Set.
3 IoT System User Group handles IoT Edge System Value Delivering.
4 IoT System Set is physical.
5 IoT System Set consists of Near End System Set and Front End System Set
6   Near End System Set is physical
7   Near End System Set exhibits NE System Dependability
8     NE System Dependability can be low or high
9     low is initial
10    high is final
11   Front End System Set is physical
12   Front End System Set exhibits FE System Dependability
13     FE System Dependability can be low or high
14     low is initial
15     high is final.
16 IoT Edge System Value Delivering requires Front End System Set and Near End System Set.
17 IoT Edge System Value Delivering changes FE System Dependability from low to high and NE System Dependability
    from low to high.

```

The OPD in Figure 4.1 and OPL in Listing 4.1 generates simultaneously in OPCAT 4.2. OPM applies *Mayer's Cognitive Assumptions* for dealing with continuous learning about the system under specification, Dori (2011). For brevity on applying *Mayer's Cognitive Assumptions* in the ISARC, dual-channel processing delivers both graphical modelling and text output during a single modelling activity.

In managing the verbose nature of the OPL, a manual adjustment occurs in the OPL by changing the auto-generated state objects such as those found in Lines 9, 10, 14, and 15. Where states appear one line after the next, a manual adjustment to concatenate the states on a single line aids in reducing the number of lines in the listing. This notation is present in subsequent OPLs where a single-page representation of the OPL is necessary.

4.1.2 System Diagram (SD1): IoT Edge System Value Delivering

An SD1 for [IoT Edge System Value Delivering](#) is zoomed-in to create the next level of detail in the system model. In line with the IoT domain structure for IoT systems, the SD1 outlines the domain structure indicated in Chapter 1, Figure 1.1 and Table 3.5 Requirement Identifier B1.1, B1.2, and B1.3. Additionally, the model references the definition of near and front end systems as seen in Yu et al. (2017), Jain and Mohapatra (2018), and Chang et al. (2019). The SD1 also references the requirements for edge devices stated in B3.1 and B3.2 in Table 3.5. The processes are shown as sequential processes; however, data acquisition can occur asynchronously to data transmitting.

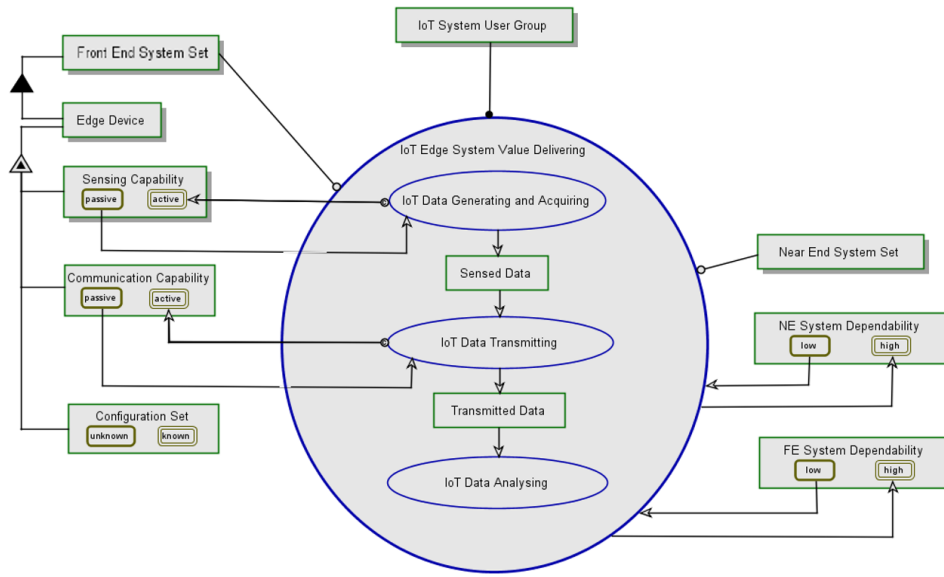


Figure 4.2: Primary Process:SD1 IoT Edge System Value Delivering

Listing 4.2: IoT System Primary Process OPL

```

1 IoT System User Group is physical.
2 IoT System User Group handles IoT Edge System Value Delivering.
3 Near End System Set is physical.
4 Front End System Set is physical.
5 Front End System Set consists of Edge Device
6   Edge Device is physical
7   Edge Device exhibits Sensing Capability, Communication Capability, and Configuration Set
8     Sensing Capability is physical
9     Sensing Capability can be passive or active
10      passive is initial active is final
11     Communication Capability can be passive or active
12      passive is initial active is final
13     Configuration Set can be unknown or known
14      unknown is initial known is final.
15 NE System Dependability can be low or high
16   low is initial high is final.
17 FE System Dependability can be low or high
18   low is initial high is final.
19 IoT Edge System Value Delivering exhibits Sensed Data and Transmitted Data.
20 IoT Edge System Value Delivering consists of IoT Data Generating and Acquiring, IoT Data Transmitting, and IoT
    Data Analysing.
21 IoT Edge System Value Delivering requires Front End System Set and Near End System Set.
22 IoT Edge System Value Delivering changes FE System Dependability from low to high and NE System Dependability
    from low to high.
23 IoT Edge System Value Delivering zooms into IoT Data Generating and Acquiring, IoT Data Transmitting, and IoT
    Data Analysing, as well as Transmitted Data and Sensed Data.
24   IoT Data Generating and Acquiring occurs if Sensing Capability is active
25   IoT Data Generating and Acquiring changes Sensing Capability from passive to active
26   IoT Data Generating and Acquiring yields Sensed Data
27   IoT Data Transmitting occurs if Communication Capability is active
28   IoT Data Transmitting changes Communication Capability from passive to active
29   IoT Data Transmitting consumes Sensed Data
30   IoT Data Transmitting yields Transmitted Data
31   IoT Data Analysing consumes Transmitted Data.

```

Once the SD and initial SD1 suitably represent the core problem, the requirements for

subsequent SDs decompose from this baseline. In its current state, Figure 4.1 and 4.2 are testable system models up to the [IoT Data Analysing](#) sub-process in SD1. These SDs adapt the ABS model example that arrives pre-installed in OPCAT 4.2. The SD and SD1 conform to core OPM symbol usage such as;

- an SD that contains a primary process and agent link. As indicated in ISO/TC 184/SC 5 (E) (2014)(pp. 24), an agent link is a procedural link that links an agent to the process it enables,
- an in-zoomed process that changes core object states including conditional links, and
- suitable SD1 instrument (enabler) links that inherit from the main SD.

A clear stipulation of the SD object and process ensures that the top-level SD does not become flooded with information that might be needed at a lower level SD1. By testing a single pre-existence parameter such as address - which indicates that addressing plays a key role in device identification IEEE (1990) - it could be viewed as being systemic rather than environmental. Identification is a primary system parameter discussed in Chapter 3 in the functions table relating to identification and management. Table 3.10 indicate these as;

- Function A1.3 [Identifying](#) changes [Thing](#) from [unidentified](#) to [identified](#),
- Function C1.4 [Remote Managing](#) changes [ED](#) from [unmanaged](#) to [managed](#), and
- Function C2.5 [Remote Managing](#) changes [GW](#) from [unmanaged](#) to [managed](#).

As indicated in IEEE (1990), the definition of an address is “*a number, character or group of characters that identifies a given device or storage location*”. This definition therefore supports an OPM object affiliation change from unknown to systemic, due to the primary functions of [Identifying](#) and [Managing](#). Table 3.6 C1.1 [Communication](#) manifests as a two-way channel requirement, and Table 3.5 B1.2 [Transmission](#) defines as the action of sending a request or notification towards a target system. The code block below applies a cyclical flow that continuously gains knowledge of an intended parameter set classification. The listing below applies the realisation method seen in Figure 3.5.

Listing 4.3: ISARC System Realisation Flow. Adapted from Figure 3.5

```

1 Identify Problem Domain.
2 Is Thing Object (Enabler or Agent) or Process?
3   If Object, then is Systemic or Environmental? OR If Process, then Can Transform or Cannot Transform?
4   If Object is Environmental AND Process Cannot Transform. Identify Next Thing.
5   If Object is Systemic AND Process Can Transform. Assign to Primary OR Secondary Function.
6 Return to Problem Domain.
```

As noted in the output of the code block, assignment to the primary function within the problem domain is vital. Other outputs define the parameters that exist outside of the control of the system - the environmental objects and processes. In a concise form, the execution of the flow enables a hard in and out of boundary parameter split. Applying this view to the *address*, *communication*, and *transmission* parameter examples, suggests the pre-existence requirements in Table 4.2. For brevity, parameters appear apart from the object construct only to describe the pre-existence concept in Table 4.2.

Table 4.2: ISARC Parameter Sample

Param. Id.	Systems Architecture Decision String
(p1)	Parameter::address , O:Address , P:Device Identifying, S:Identification Unknown or S:Identification Known
(p2)	Parameter::communication , O:Device , P:Device Connection Establishing, S:Device Not Connected or S:Device Connected
(p3)	Parameter::transmission , O:User or Control Data, P:Device Sending, S:Transmitted Data Failure or S:Transmitted Data Success
...	...

The level of abstraction exhibited in simple parameters can be too high to show system value. The question that subsequent sections on system diagrams will explore is balancing the abstraction and maintaining a suitable level of model completeness that increases model fidelity. Figures 4.1 and 4.2 show a series of high level system purpose and primary process objects that validate successfully in OPCAT 4.2. Due to a large possible requirement and variation list in the parameter Tables 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9, a set of candidate parameters acts as the catalyst to ongoing model refinement. Selecting from a finite list of parameters creates a clear and relevant system purpose. This suggestion informs via “good-model” quality requirements indicated in the verification and validation attributes for model design in Cameron et al. (2017).

4.1.3 System Diagram (SD1.1): IoT Edge Data Transmitting

Figure 4.3 outlines an SD 1.1 for the transmitting functions in an IoT-Ce system. SD1.1 presents an in-zoomed process showing data transmission as a sequential process; however, it may be asynchronous in its use context. The pre-existence suggestion is that the notification to an external system requires the transmission of data. However, there is no requirement for *transmission* to occur between *Front-end System Sets* when *data generating* and *acquiring* occurs. Devices can sense data without using network services. The information about the device *address* is a pre-existence requirement for *transmission*, but not for handling of address registers internal to the device. The definition of an *address* is, however, “*a number, character or group of characters that identifies a given device or storage location*”, IEEE (1990).

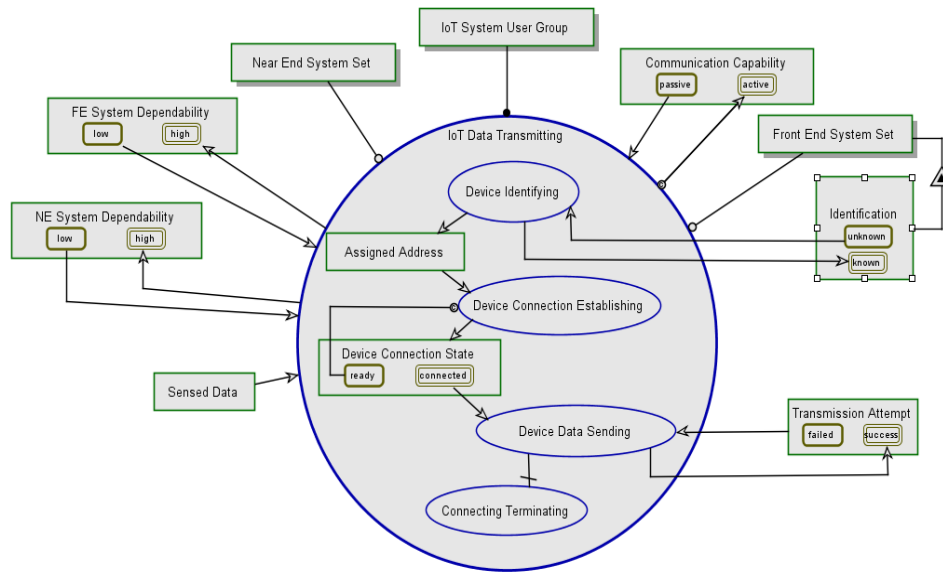


Figure 4.3: In Zoomed:SD1.1 IoT Data Transmitting

Listing 4.4: IoT Data Transmitting SD1.1

```

1 IoT System User Group is physical.
2 IoT System User Group handles IoT Data Transmitting.
3 Near End System Set is physical.
4 Front End System Set is physical.
5 Front End System Set exhibits Identification
6   Identification can be unknown or known
7     unknown is initial known is final.
8 NE System Dependability can be low or high
9   low is initial high is final.
10 FE System Dependability can be low or high
11   low is initial high is final
12 Communication Capability can be passive or active}. (p2)
13   passive is initial active is final.
14 Transmission Attempt can be failed or success. (p3)
15   failed is initial success is final.
16 IoT Data Transmitting exhibits Assigned Address and Device Connection State. (p1)
17 IoT Data Transmitting consists of Device Identifying, Device Connection Establishing, Device Data Sending, and
18   Connecting Terminating.
19 IoT Data Transmitting occurs if Communication Capability is active. (p3)
20 IoT Data Transmitting requires Near End System Set and Front End System Set.
21 IoT Data Transmitting changes FE System Dependability from low to high, Communication Capability from passive
22   to active, and NE System Dependability from low to high.
23 IoT Data Transmitting consumes Sensed Data.
24 IoT Data Transmitting zooms into Device Identifying, Device Connection Establishing, Device Data Sending, and
25   Connecting Terminating, as well as Device Connection State and Assigned Address
26   Device Connection State can be ready or connected
27     ready is initial connected is final
28   Device Identifying changes Identification from unknown to known
29   Device Identifying yields Assigned Address. (p1)
30   Device Connection Establishing occurs if Device Connection State is ready
31   Device Connection Establishing consumes Assigned Address
32   Device Connection Establishing yields Device Connection State
33   Device Data Sending consumes Transmission Attempt and connected Device Connection State
34   Device Data Sending yields success Transmission Attempt
35   Device Data Sending triggers Connecting Terminating when it lasts more than 5 seconds. (Expanded Constraint)

```

The OPL in the code block represents a view aligned to the primary parameter sample in Ta-

ble 4.2. Lines 12, 14, 16, 18, 26, and 32 further clarify some of the main system requirements between the content of all other objects and processes in the OPL. The definition-to-OPM filter produces from the samples in Table 4.2 yields a suitable model parameter identification approach when considering input requirements for model outputs such as those seen in Figure 4.3.

In summary, the extraction of the parameters from the main IoT body of work handled in preceding chapters produces a parameter set contained in Tables 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9. These tables reduce in Table 4.2. The parameter selection sample in Table 4.2 is and output using the ISARC system realisation flow. The parameter selection could occur randomly; however, the ISARC intends to drive SD to SDx.x.x modelling questions that maintain clarity in the system purpose and distributes complete system component placement in the correct levelled SD. [IoT Edge System Value Delivering](#) represents a requirement for a [Front-end](#) and [Near-end system set](#). Additionally, the known IoT-Ce domain requirements of [Generating](#), [Acquiring](#), and [Transmitting](#) exists as processes that support the primary goal of delivering the value of increased dependability. Additionally, [Addressing](#) appears as a precursor to [Transmitting](#). The parameter samples found in Table 4.2 produces a series of candidate objects, processes, and states for testing individual configurations - an arrangement of variables. A set of desired sample processes and instruments required for those processes appear in the same table. The main process models a series of system questions such as system representation, decomposition, and aggregation seen in Figure 4.3. Finally, although not required entirely, the SD1.1 is refined to ensure that the last process positions to execute a non-blocking behaviour that enforces a time-bound constraint. This characteristic appears in line 32 of the SD1.1 OPL.

The next section presents the handling of configuration and control against the SD, SD1, and SD1.1 as supporting objects and processes in OPM.

4.2 Supporting Objects & Processes

Supporting *Things* (Objects and Processes) enable the system's primary purpose by supporting the primary function, such as the primary function seen in Figure 3.5. Figure 3.5 presents a cyclical flow that filters through a torrent of parameters and systems diagram content. The flow cycles through parameters until a primary candidate emerges. However, supporting objects and processes can emerge if the object is neither environmental nor unable to assist in an object state change. Supporting objects could physically or informatically support the primary system function of data transmitting for increased dependability. The

definition of supporting functions noted in Crawley et al. (2016)(pp.126) is the form and function for “one or two layers away from the value delivery pathway that is necessary to support the value delivery functions and their instruments”.

The primary purpose in the SD in Figure 4.1 states as; stakeholder value delivering through increased **Front-end System Set** dependability. Essentially, increases in **Front-end System Set** dependability delivers the actual system value to the IoT system stakeholder group. More directly, increases in **Front-end System Set** dependability delivers value to a mobile device user (IoT System User Group). The remaining objects positioned by the parameter extract in Table 3.10 represent a class of supporting objects. These objects support increased dependability via self-adaptation in autonomic control.

- F1.1 Monitor = **Monitoring**
- F1.2 Analyse = **Analysing**
- F1.3 Plan = **Planning**
- F1.4 Execute = **Executing**
- F1.5 Knowledge = **Knowledge Handling**, **Solution Topology (ST) Knowledge**, **Policy (PO) Knowledge**, **Problem Determination (PD) Knowledge**

Additional examples of the ISARC generated OPM string for MAPE-K appear in Table 4.3. The decomposition and subsequent function aligns to the definition of the parameters extracted from IBM (2005)(pp.16). This text and supporting texts already propose that

Table 4.3: ISARC Supporting Parameter Sample

Systems Architecture Decision String
Parameter::monitor, O:Managed Element Configuration, P:Managed Element Configuration Retrieving, S:Managed Element Configuration Unknown or S:Managed Element Configuration Known
Parameter::analyse, O:Situation, P:Situation Awareness Handling, S:Situation Correlated or S:Situation Not Correlated
Parameter::plan, O:Action, P:Plan Constructing, S:Goal/Object Action Plan Achieved or S:Goal/Object Action Plan Not Achieved
Parameter::execute, O:Plan, P:Plan Executing, S:Plan Executed or S:Plan Not Executed
Parameter::knowledge, O:Configuration, P:Configuration Identifying, S:ST Knowledge Unknown or S:ST Knowledge Known S:PO Knowledge Unknown or S:PO Knowledge Known, S:PD Knowledge Unknown or S:PD Knowledge Known,

the visibility of state increases the ability to control system value. Thus, state extraction outputs align with the knowledge of the configuration of a **Front-end System Set** at the end of the monitoring process. It has also been noted that the knowledge of state in configurable systems is necessary for execution of determined actions, Feiler and Li (1998), Cheng (2008)

and Swanson et al. (2014).

The supporting objects and process section aims to expand what already exists for IoT-Ce SDs in sections 4.1.1, 4.1.2, and 4.1.3. Thus, the modelling of the monitor, analyse, plan, execute, and knowledge parameters for the MAPE-K framework will not present any new knowledge at this point about the SD1.1. However, applying control concepts to a concrete control problem represents the applicability of using OPM to implement different control characteristics for applications in a real-world example. A novel control system overview using OPM is already covered in Dori (2002), shown in Appendix A.4. However, the supporting objects for the ISARC apply a control strategy that applies closer to the IoT-Ce.

Self-* properties support adaptive systems via closed-loop control mechanisms, as indicated in Table 2.5. The basis of the ISARC parameter extraction for the system elements that model the control constraints in a closed-loop system appears in the constraints section of section 1.7. These constraints adapt from Janert (2013). Firstly, the original, non-OPM-based, control constraint in Janert (2013) appear as;

1. a variable control initiation measurement, measurement time period, target control position and time step,
2. a set of standard control gains are applied periodically on the time step,
3. requests not handled are treated as failed requests,
4. no queued requests to be processed during or after the measurement time period,
5. a finite number of request and responses and
6. a random load intensity over the measurement time period.

Then an OPM extraction defines the **Object**, **Process**, and **Attribute (A)** equivalence as;

1. **Control Initiation Measurement**, **Measurement Time Period**, **Target Control Position** and **Time Step**, exhibits **Variability(A)**,
2. **Standard Control Gain Setting** applied periodically on the **Time Step**,
3. **Request** not handled invokes **Failed State Notifying**,
4. **Queued requests** triggers **Queued Request Purging** when it occurs during or after **Measurement Time Period**,

5. **GW Request** and **GW Response** and **ED Request** and **ED Response** exhibits a **Finite Count(A)** and
6. **Traffic Load Intensity** exhibits **Random Load(A)** and is constrained by a **Measurement Time Period**.

The additional identifier, attribute (A), is included in the analysis of the control problem above as some objects could exhibit a discrete choice behaviour. *Attributes* are objects that indicate the characteristics of the main object description. ISO/TC 184/SC 5 (E) (2014)(3.5) defines an attribute as an “object that characterises a thing other than itself”. In the control problem above, measurements, steps, and control positions could be either variable or fixed. The problem statement for the ISARC calls for variability as opposed to fixed measurements for managing control outputs as architecture decisions are largely non-deterministic.

In the definition-to-OPM extraction list item 6, a **Random Load** characterises the **Load** object. The load intensity changes can be summarised to be in line with similar views on random ED placement, cited in Bera et al. (2016). Similarly, the number of requests in the control constraint is finite, as opposed to infinite. Thus, the **Count** object exhibits a **Finite** characteristic. The initial ISARC supporting function for the control problem is labelled **IoT System Control Handling** which consumes an **IoT System Controller** in Figure 4.4. The attachment of the IoT control handling process to the IoT system controller is unique to the ISARC.

A subsequent section deals with the placement of a supporting control function within the primary systems architecture boundary. Note, the flexibility of switching *Things* between systemic or environmental occurs by changing the *Affiliation*, ISO/TC 184/SC 5 (E) (2014)(section 7.3.3). Before pre-empting the value that system control offers to an IoT-Ce, the controller is placed as an environmental function. The **IoT System Control Handling** process attaches to the **IoT Edge System Value Delivering** function. By using environmental affiliation, the control function impacts the system but not the inverse. The **IoT System Control Handling** process and **IoT System Controller** object is shown to contain a dashed outline. Dashed outlines indicate environmental placement in OPM.

Figure 4.4 attaches the new process and object as environmental objects, however it is modelled to affect the **Front-end System Set** dependability. The OPL is automatically updated to show the insertion of the new object and process relationships.

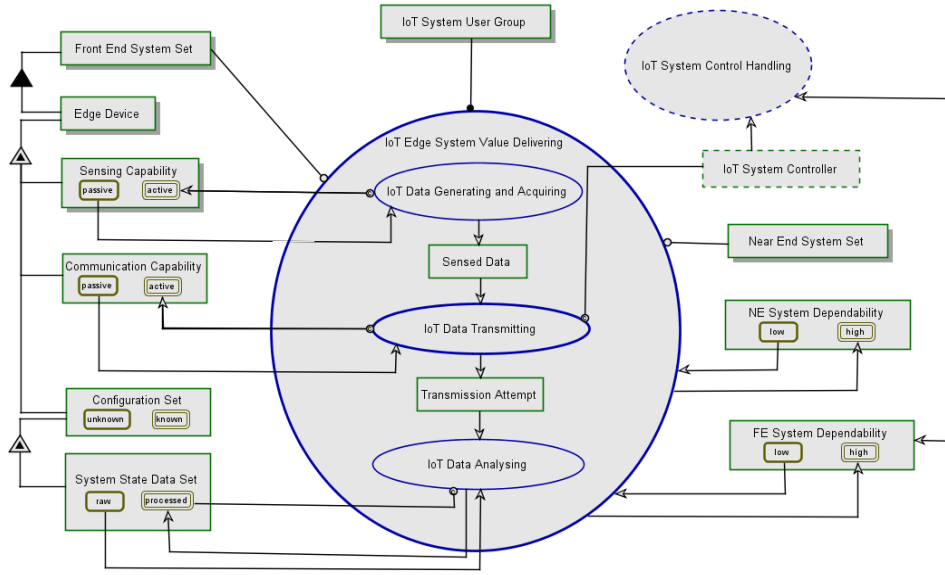


Figure 4.4: SD1 Extension for System Value Environmental Control Configuration

Listing 4.5: Supporting Process:SD1 IoT-Ce Value Delivering Updated Configuration

```

1 ...
2 ...
3 IoT System Control Handling is environmental.
4 IoT System Control Handling affects FE System Dependability.
5 IoT System Control Handling consumes IoT System Controller.
6 IoT Edge System Value Delivering exhibits Sensed Data and Transmission Attempt.
7 IoT Edge System Value Delivering consists of IoT Data Generating and Acquiring, IoT Data Transmitting, and IoT
  Data Analysing.
8 IoT Edge System Value Delivering requires Front End System Set, and Near End System Set.
9 ...
10 ...
11 IoT Data Transmitting occurs if IoT System Controller is in existent and Communication Capability is active.
12 ...
13 ...

```

In Figure 4.4, the controller is attached to the **IoT Data Transmitting** process. This SD1 placement is preferred due to the nature of the control problem identified in Janert (2013). Janert's constraint problem resembles a one-way transmission mechanism as opposed to a bidirectional transmission pattern. Note, that Figure 4.4 maintains the original model structure adapted from Figure 4.2. The ability to test controller impacts is possible by attaching the controller within one or more layers in the SD1. The handling of control at a suitably abstracted level also places within the top-level SD instead of being attached to either **IoT Data Generating & Acquiring**, **IoT Data Transmitting**, or **IoT Data Analysing** in the SD1. The impact of control can, therefore, specify coverage across all domains of interest in the IoT-Ce. Hence, the control functions would locate as a process inside the **IoT Edge System Value Delivering** boundary and alongside the three domains of interest in Figure 4.4.

Figure 4.5 outlines an extended SD 1.1 for the transmitting function. The model zooms in from SD1 to SD1.1. Figure 4.5 applies external control in an IoT-Ce system by zooming into the transmission function, and attaching to the [Device Connection Establishing](#) process. To recall, the Systems Architecture Decision String in Table 4.2 appears as;

- (p2) - **Parameter**::communication, **O**:Device, **P**:Device Connection Establishing, **S**:Device Not Connected or **S**:Device Connected

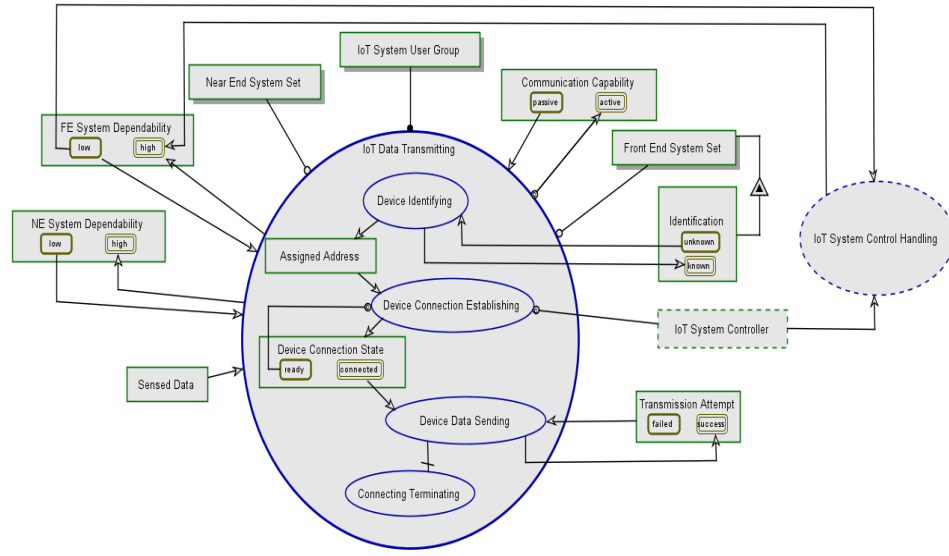


Figure 4.5: In Zoomed:SD1.1 IoT Data Transmitting Environmental Control Candidate

Figure 4.5 applies a direct SD1.1 sub process requirement for connection establishment, and attaches the control requirement directly to this sub-process. The [IoT System Control Handling](#) process direct changes the [Front-end System Set](#) from [low](#) to [high](#). The in-zoomed process immediately exposes the parameter-based sub-processes; [Device Identifying](#), [Device Connection Establishing](#), [Device Data Sending](#), and [Connecting Terminating](#). The latter process applies to the notion of ensuring that no stalled connections are experienced in the process, making the return journey of the process non-blocking.

A detailed structural view of the control parameters by Janert (2013) expands the [IoT System Control Handling](#) process. The in-zoomed [IoT System Control Handling](#) process takes the liberty of providing more information in the unfolded controller SD. This view is contrary to the OPM view on balancing completeness and clarity characteristics in the model's representation, Dori (2011). The supporting object SDs provides a critical mass of objects and processes that visually represent a suitable simulation-ready model while

minimising hierarchy switching. Dori provides a similar verbose structure for his controller overview, Dori (2002)(pp.393). The definition-to-OPM extraction equivalent, proposed by Janert (2013), is unfolded in Figure 4.6. The OPM equivalent structure adapts the main control problem shown by Janert’s original requirements. These are; (1) variable control action and settings, (2) control gain invoked from the main control handling process, (3) failed request notification, (4) queued request purging, (5) time-bound, and (6) random load behaviour.

Figure 4.6 is the primary model link between the environmental control system, the Android

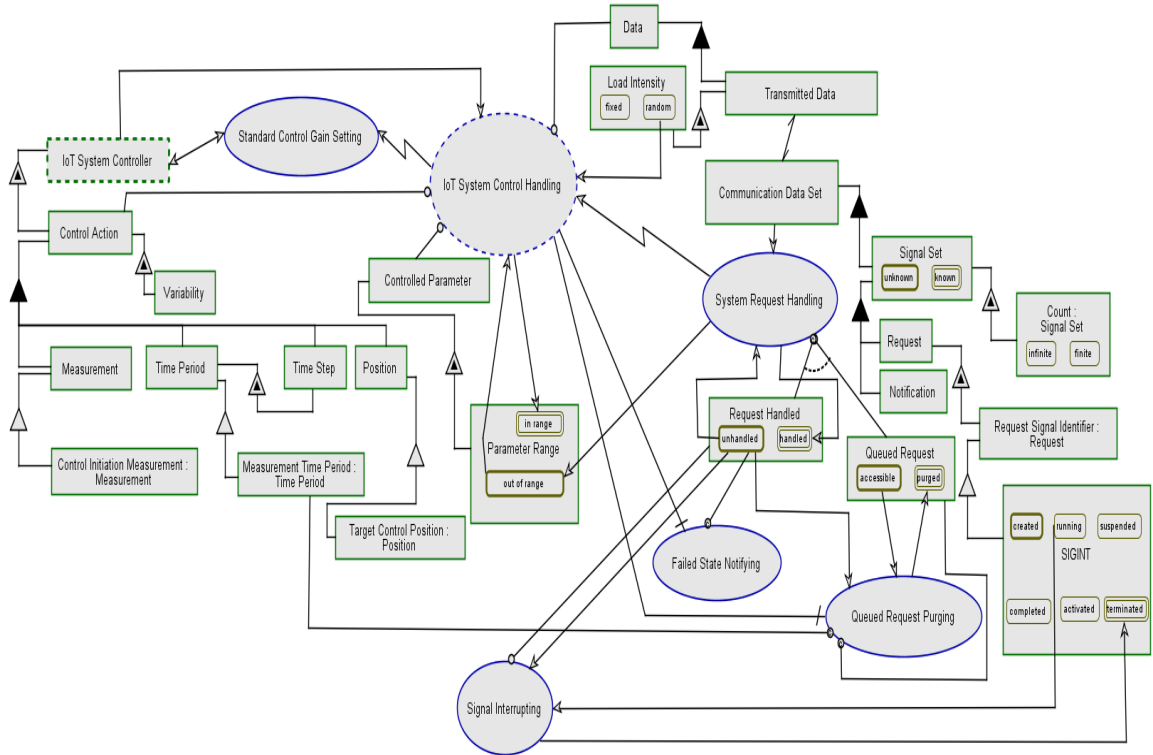


Figure 4.6: Unfolded Structure:IoT System Controller Candidate

NodeJS signal constants exposed via the **Front-end System Set** mentioned in section 3.4.2, and process state advancement in section 2.2.3.1. The system under control aims to maintain a system attribute from **out of range** to **in range**. Dori (2002)(pp.391) indicates these ranges as too high, too low, or acceptable. Figure 4.6 models a detailed interaction between control and the abstracted constant SIGINT. SIGINT is an Android system control that sends an interrupt to a system process, effectively halting a running process. A simple SD expansion to the more complex unfolded structure shows that single model requirements traverse from top-level system purpose diagrams with fewer OPM things towards more complex interactions. Having modelled an interacting process, several refinements could be applied to Figure

4.6 to maintain model clarity and completeness. More verbose model content increases the ability of Figure 4.6 to be validated directly in OPCAT 4.2. However, this increase creates possible imbalances in clarity and completeness model parameters stipulated by Dori.

Listing 4.6: Unfolded Structure:IoT System Controller Candidate

```

1 Communication Data Set consists of Signal Set.
2   Signal Set can be unknown or known.
3     unknown is initial.
4     known is final.
5   Signal Set exhibits Count.
6     Count can be infinite or finite.
7   Signal Set consists of Request and Notification.
8     Request exhibits Request Signal Identifier.
9       Request Signal Identifier is of type Request.
10 Communication Data Set and Transmitted Data are equivalent.
11 Request Handled can be unhandled or handled.
12   unhandled is initial.
13   handled is final.
14 Queued Request can be accessible or purged.
15   accessible is initial.
16   purged is final.
17 Controlled Parameter exhibits Parameter Range.
18   Parameter Range can be out of range by default or in range.
19     out of range is initial.
20     in range is final.
21 Data consists of Transmitted Data.
22   Transmitted Data exhibits Load Intensity.
23     Load Intensity can be fixed or random.
24 IoT System Controller is environmental.
25 IoT System Controller exhibits Control Action.
26   Control Action exhibits Variability.
27   Control Action consists of Measurement, Time Period, Position, and Time Step.
28     Time Period exhibits Time Step.
29 Control Initiation Measurement is a Measurement.
30 Target Control Position is a Position.
31 Measurement Time Period is a Time Period.
32 SIGINT is a Request Signal Identifier.
33 SIGINT can be created, running, suspended, completed, activated, or terminated.
34   created is initial.
35   terminated is final.
36 IoT System Control Handling is environmental.
37 IoT System Control Handling requires Data, Controlled Parameter, and Control Action.
38 IoT System Control Handling changes Parameter Range from out of range to in range.
39 IoT System Control Handling consumes random Load Intensity and IoT System Controller.
40 IoT System Control Handling triggers Queued Request Purging and Failed State Notifying when it lasts more than
    5 seconds.
41 IoT System Control Handling invokes Standard Control Gain Setting.
42 Standard Control Gain Setting affects IoT System Controller.
43 Failed State Notifying occurs if Request Handled is unhandled.
44 System Request Handling occurs if either Queued Request is in existent or Request Handled is in existent.
45 System Request Handling changes Request Handled from unhandled to handled.
46 System Request Handling consumes Communication Data Set.
47 System Request Handling yields out of range Parameter Range.
48 System Request Handling invokes IoT System Control Handling.
49 Queued Request Purging occurs if Queued Request is in existent and Measurement Time Period is in existent.
50 Queued Request Purging changes Queued Request from accessible to purged.
51 Queued Request Purging consumes unhandled Request Handled.
52 Signal Interrupting requires Request Handled.
53 Signal Interrupting changes SIGINT from running to terminated.
54 Signal Interrupting consumes unhandled Request Handled.

```

A system test of the flow paths in Figure 4.6 is handled in Chapter 5 - Model Validat-

tion & Evidence. The changes to the object state in the main execution path occur from both internal systems and external system processes. For example, handling queued request purging requires both the existence of an accessible **queued request** and the existence of a **measurement time-period** as indicated in requirement 4 of the main control problem in the OPM equivalent structure. This requirement is shown in line 49 of the Primary Unfolded Structure:IoT System Controller Candidate OPL. Additionally, as per the concrete control requirements in Janert (2013), the request is changed from an **accessible** to a **purged** state. Finally, a control constraint shows that the purging process consumes an **unhandled request**.

The supporting objects and processes summarised in this section create the basis of an environmental IoT control system that immediately implements six core requirements. These core requirements represent a concrete control problem. The requirements enable creating a single subsystem attached to either SD, SD1, or SD1.1. Once attached, it unfolds to lower level handling of signalling constant change that still meets OPM system validation requirements. Subsequent sections will produce system object lifespans to prove the ability of the model to meet validations under different model questions. This section aims to present a clear separation of concerns in the OPDs produced. For primary purpose and processes in the ISARC, a series of parameters are shown in earlier tables that can decompose from SD to SD1 to SD1.1, towards lower-level SDn.n.n nodes. For supporting objects and processes, A single dependability impacting subsystem model is present—the initial argument positions this subsystem as existing outside of the system boundary as an environmental subsystem. This model guidance is to maintain strict impact boundaries. Inputs for the primary and supporting model sets apply the definition-to-OPM extraction approach.

4.3 System Interfaces & States

System interfacing occurs at system boundaries. The interaction between subsystems is the expression point where system value increases or decreases. How subsystems or components interface is a result of conditions that exist at the system boundary. Several views on interfacing stipulate that things connect at a system boundary, Kopetz (2011), ISO/TC 184/SC 5 (E) (2014) and Mordecai and Dori (2017a). Hence, the CPS interfacing theory suggests the abstraction of core components away from their physical implementation, Lohstroh and Lee (2015). Operating System control for **Front-end System Set** instances provides a common set of system processes suitable for bounded control. In section 3.4.2, high-level programming application libraries abstract the operating system to provide greater open access integration compared to serial connectivity for devices. This decoupled approach supports the necessary hardware abstraction needed in the systems architecture for state-of-the-art control in

the IoT-Ce. A common theme cited is the access of **Front-end System Set** states through several architectural layers, Linux (2018). The systems architecture is, of course, the primary guidance in the ISARC. The interest lies in which system value drivers generalise the abstraction of computer-based subsystem or component sets. Figure 4.7 produces a sum-

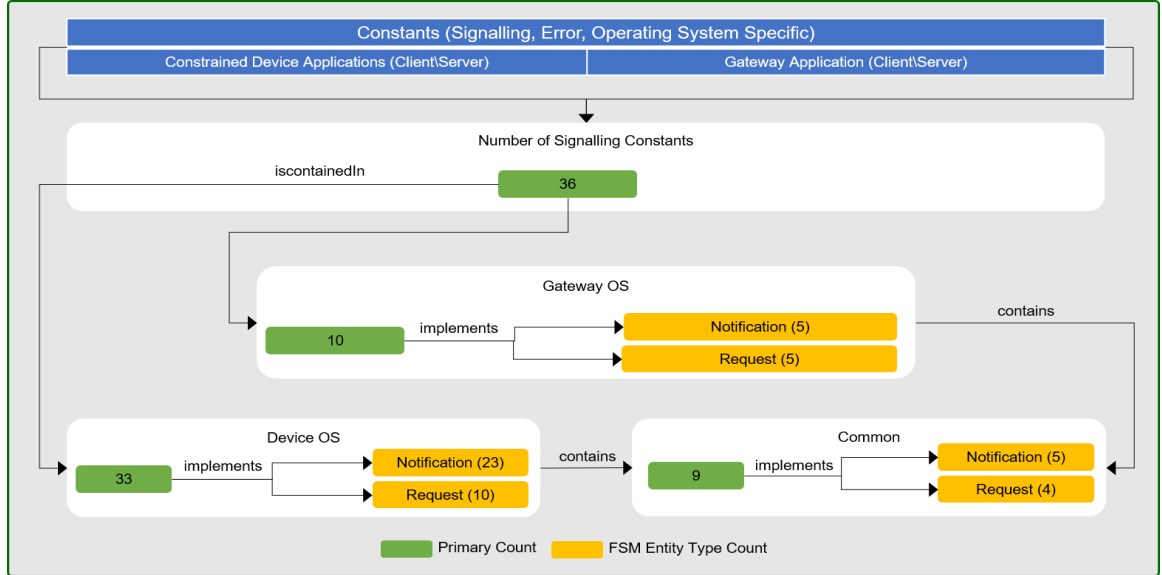


Figure 4.7: Analysis of Node JS Signalling Constants extracted from Linux (2018)

mary of the proposed **Front-end System Set** common signalling constants consisting of the 36 known signalling requirements found in Linux (2018). Figure 4.7 outlines two essential interaction classes; *Notification* and *Request*. These were discussed in the parameter Table 3.7. Figure 4.7 produces a summary of the proposed ED/GW common signalling constants. Additionally, Figure 4.7 outlines two essential interaction classes; *Notification* and *Request*. The interaction classes appear in Table 3.7 as OPM-extracted parameters.

The analysis of the allowable Android notification and request processes in Linux (2018) further indicates that particular notify actions initiate a feed-forward loop with possible further action to correct an anticipated event. Examples of these are *SIGXCPU* and *SIGXFSZ*, which communicates a breach to the central processing unit (CPU) and filesize limits, respectively. An internal **Front-end System Set** process to initiate a guard condition such as *SIGUSR1* or *SIGUSR2* could, therefore, initiate to ensure that *SIGSTOP* does not cause a halt of a **Front-end System Set** process. A comprehensive list of the signalling notification and response descriptions that exists in the NodeJS OS library is shown in Table 4.4 and 4.5.

The signalling constructs in Table 4.4 provide a less onerous process on the subsystem

Table 4.4: Node JS Signalling Notify Constants extracted from Android OS Constants in Linux (2018)

Notification	Description
SIGHUP	Sent to indicate when a controlling terminal is closed or a parent process exits.
SIGILL	Sent to a process to notify that it has attempted to perform an illegal, malformed, unknown, or privileged instruction.
SIGTRAP	Sent to a process when an exception has occurred.
SIGBUS	Sent to a process to notify that it has caused a bus error.
SIGFPE	Sent to a process to notify that it has performed an illegal arithmetic operation.
SIGSEGV	Sent to a process to notify of a segmentation fault.
SIGPIPE	Sent to a process when it has attempted to write to a disconnected pipe.
SIGALRM	Sent to a process when a system timer elapses.
SIGCHLD	Sent to a process when a child process terminates.
SIGSTKFLT	Sent to a process to indicate a stack fault on a coprocessor.
SIGTTIN	Sent to a process when it reads from the TTY while in the background.
SIGTTOU	Sent to a process when it writes to the TTY while in the background.
SIGURG	Sent to a process when a socket has urgent data to read.
SIGXCPU	Sent to a process when it has exceeded its limit on CPU usage.
SIGXFSZ	Sent to a process when it grows a file larger than the maximum allowed.
SIGVTALRM	Sent to a process when a virtual timer has elapsed.
SIGPROF	Sent to a process when a system timer has elapsed.
SIGWINCH	Sent to a process when the controlling terminal has changed its size.
SIGIO	Sent to a process when I/O is available.
SIGPOLL	Synonym for SIGIO
SIGLOST	Sent to a process when a file lock has been lost.
SIGPWR	Sent to a process to notify of a power failure.
SIGINFO	Synonym for SIGPWR
SIGSYS	Sent to a process to notify of a bad argument.
SIGUNUSED	Synonym for SIGSYS

or components with which it communicates as these states can be accessed directly and are seamless aware of the state of the **Front-end System Set**. The descriptions in Table 4.4 entail a notification but not necessarily the subsequent action event that needs a process to execute. For example, *SIGTRAP* defines the notification sent to a process indicating that an exception has occurred. By definition of the error to fault escalation process seen in Figure A.1, if this exception handles within acceptable limits, it would be classified as an exception handler which does not entirely impact the reliability of the component. Thus, the system could remain in a functioning state (state:1) when the *SIGTRAP* transmits to a delinquent process.

A structurally unfolded OPD is shown in Figure 4.8. The associated OPL is also shown. Each **Notification State Identifier** can be further unfolded to reveal a more granular set of compositions. Crawley et al. (2016) refer to this as form decomposition activities. As indicated in the unfolded structure for the **IoT System Controller** in Figure 4.6, the object can hold several states such as **created**, **running**, **completed**, **suspended**, **activated** or **terminated**. These states adapt from the process transition baseline included in Table 3.8.

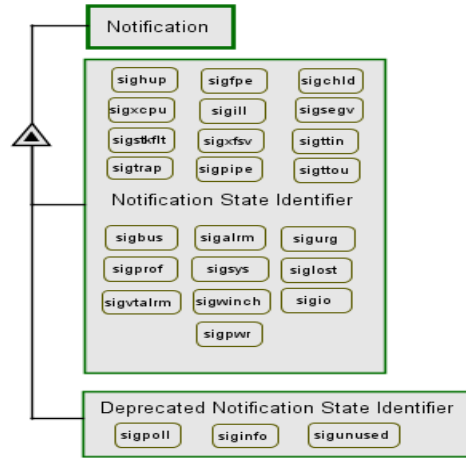


Figure 4.8: OPD Unfolded Constant Notify Structure

Listing 4.7: OPL Unfolded Constant Notify Structure

```

1 Notification exhibits Notification State Identifier and Deprecated
2 Notification State Identifier.
3 Notification State Identifier can be sighup, sigill, sigtrap, sigbus, sigfpe, sigsegv, sigpipe, sigalrm,
  sighld, sigstkflt, sigttin, sigttou, sigurg, sigxcpu, sigxfsv, sigvtalrm, sigprof, sigwinch, sigio,
  siglost, sigpwr, or sigsys.
4 Deprecated Notification State Identifier can be sigpoll, siginfo, or sigunused.

```

The ISARC thus models the awareness of each notification state in Table 4.4 has a single purpose in as each notification is “Sent to indicate” a notification action that executes during runtime, allowing the request application - sender of the notification - express knowledge of the event that occurred before the notification is triggered. The **event** occurs before the **notifying** process. For example, for *SIGHUP* notification is sent when a **controlling terminal** is **closed**, and thus the process is **Closed Terminal Notifying**, and consequently the function; **Closed Terminal Notifying** yields **notification of controlling terminal state change** from **open** to **closed**.

A state change request in Linux (2018) represents a firm instruction to perform an action using NodeJS on Android. Examples of this in the constant signalling list are *SIGINT* which requests an immediate interrupt of a process using a Ctrl+C combination. Similarly, *SIGABRT* and *SIGTSTP* which requests a process abort and stop, respectively. The process descriptions in the signalling definitions in Tables 4.4 and 4.5 lean towards classical process modelling where the process descriptions reference inputs and outputs. For example, the NodeJS request SIGINT is sent to a component to indicate an interrupt. Thus, Ctrl+C is the output sent to a running component’s input to interrupt the process. Once the system interrupt reaches the **Front-end System Set** the NodeJS server becomes inaccessible, and hence no **transmission** can occur towards and away from the **Front-end System Set** . Thus,

Table 4.5: Node JS Signalling Request Constants extracted from Android OS Constants on Linux (2018)

Request	Description
SIGINT	Sent to indicate when a user wishes to interrupt a process ((Ctrl+C)).
SIGQUIT	Sent to indicate when a user wishes to terminate a process and perform a core dump.
SIGABRT	Sent to a process to request that it abort.
SIGIOT	Synonym for SIGABRT
SIGKILL	Sent to a process to terminate it immediately.
SIGUSR1	Sent to a process to identify user-defined conditions.
SIGUSR2	
SIGTERM	Sent to a process to request termination.
SIGCONT	Sent to instruct the operating system to continue a paused process.
SIGSTOP	Sent to instruct the operating system to halt a process.
SIGTSTP	Sent to a process to request it to stop.
SIGBREAK	Sent to indicate when a user wishes to interrupt a process.

creating a break in communication and decreasing dependability. Therefore, a parameter set that increases dependability by offloading current device states to a component not always susceptible to a Ctrl+C interrupt is viewed as a pre-emptive architecture decision that can increase state-awareness. The OPD and OPL show the OPM-aligned visual and text formatting.

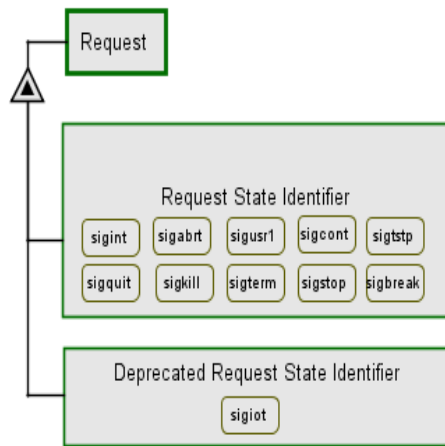


Figure 4.9: OPD Unfolded Constant Request Structure

Listing 4.8: OPL Unfolded Constant Request Structure

- 1 Request exhibits Request State Identifier **and** Deprecated Request State Identifier.
- 2 Request State Identifier can be sigint, sigquit, sigabrt, sigkill, sigusr1, sigterm, sigcont, sigstop, sigtstp, **or** sigbreak.
- 3 Deprecated Request State Identifier **is** sigiot.

The specification of state certainly expands from the functions supplied in the requirement set for IoT-Ce, to broader generality. Dori (2002)(pp.316) explores transient state management using a knowledge base for state process handling. The additional processes

that manage transient states are mentioned as; Supporting, Holding, Maintaining, Keeping, Staying, Waiting, Prolonging, Delaying, Occupying, Persisting, Including, Containing, Continuing, Enclosing, Fastening, Connecting, Postponing, Dragging, Storing, Owning, Restraining, Drawing, Attracting and Remaining. A suitably sized majority of these could impact the value IoT Edge System Value Delivering, primary function sub process of Device Connection Establishing, and supporting function pair of IoT System Control Handling process and IoT System Controller. For example, in the case of;

- **Parameter::monitor**, **O**:Managed Element Configuration, **P**:Managed Element Configuration Retrieving, **S**:Managed Element Configuration unknown or **S**:Managed Element Configuration known

If the waiting process state was used to specify a change to the **Parameter::monitor** definition, it could support the transition of configuration knowledge state from unknown to standby to known, thus increasing the possibility of integrity of runtime configuration knowledge in the detection process. A standard set of configuration state transitions in TC-SMG (1996)(pp.14) are; Alarm, Procedural, Availability, Control, Standby and Unknown.

4.4 Runtime Configuration Variation

As shown in previous chapters, the variation in system parameters' arrangement can be vast even when simple questions test against a systems architecture. One of the reasons it is so vast is the increase of IoT coverage without a requisite systems approach. Questions as simple as; Is device addressing (address) important in device identification? If so, why? What else needs to exist before addressing can occur? Can decisions relating to those crucial objects and states exist in a boundary such that it contains only the important architectural decisions? How can system value be realised while ensuring efficiency in the decision process? Does the system require a controller, and if so, what type of control is needed? What are the states that such a controller should manage? There are a number of state space combinations in the vast design, configuration, and decision space from the preceding chapters. The ISARC response to these is several fact-based definitions that are amenable to receive quantitative priorities, decision string content, and variable testing for truth-based outputs. These IoT-Ce requirement parameters are found in Tables 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9.

In the ISARC, definitions from Table 4.5 such as "SIGINT is sent to indicate when a user wishes to interrupt a process..." defines the existence of an object state such as Process

Active, or any state after **Process Created** but before **Process Terminated**. The process transitions is a primary process state-space extracted from Reichert and Weber (2012)(pp.34). The creation of a finite set of system parameters with tables such as Table 4.4 and Table 4.5, further provides a well-bounded, abstract, and suitably well covered qualitative set of decision parameters for both IoT-Ce system purpose and system function. The addition of detail from the process transitions extracted from Reichert and Weber (2012)(pp.34) closes the abstraction of processes in a system by binding the possible process transitions into a finite state space.

Figure 4.10 presents a graphical representation of the possible combination of states and

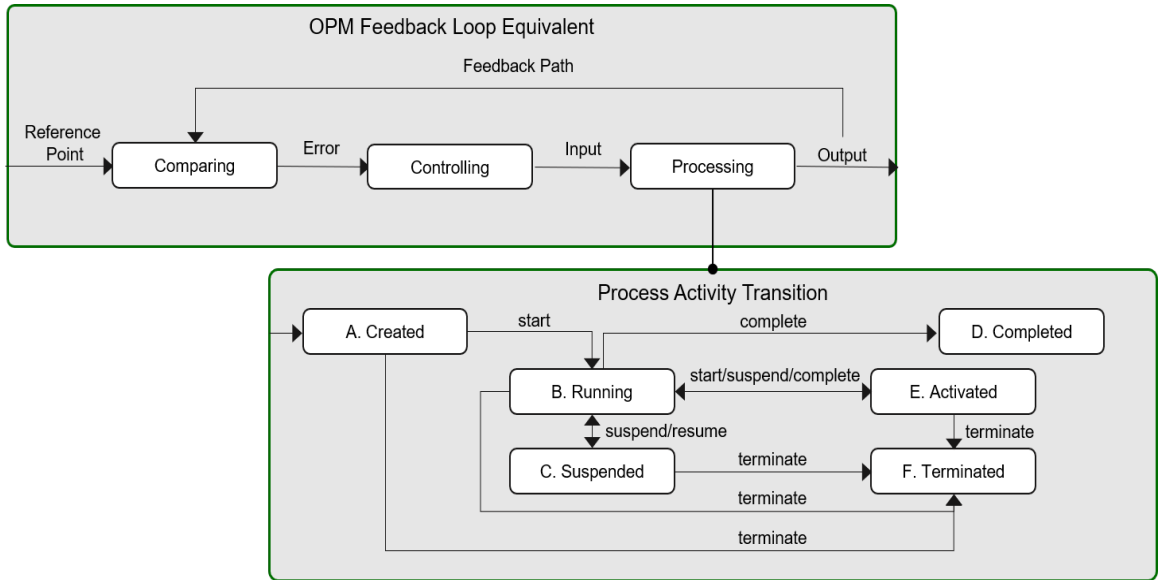


Figure 4.10: Gerund Feedback Loop Process State Transition. Adapted from Reichert and Weber (2012)

state transitions of a process instance adapted from Reichert and Weber (2012)(pp.34). As an overlay, Figure 4.10 contains the actual unmodified process transition steps included in a version of a simple closed-loop feedback system from Figure A.6 in Appendix A. The controller structure attaches as an environmental object. The ISARC does not mandate the use of a controller, but it does exhibit a functional PI controller as part of the model proof. The ISARC controller proof tests in model form and also appears in Python simulations for the ISARC.

The process included in the IoT-Ce system under control initially contains the entire process life cycle outlined in Reichert and Weber (2012)(pp.34). Thus, a **Process Handler** object is a process component within its subsystem boundary. The controller output interconnection -

which is also a controlled system input - is dealt with using an externalised control strategy instead of a strategy where the controller exists within the subsystem boundary. This strategy is an example of a configuration variation that tests for efficiencies and system impact. The effect of this specific variation is the negation of a network delay on the controlled **Front-end System Set** as the **Front-end System Set** internal control process does not extend to another system. This strategy is applicable in interactions where no hard constraint exists for continuous **communication**. Hence, inter **Front-end System Set communication** is not a pre-existence criteria for system control. An example of this is implementing asynchronous communication in non-blocking architectures in the section on Interfaces and State, section 4.3. An alternative configuration could be to place the controller outside the subsystem boundary within a spatial and temporal reach that lessens the communication latency and reduces in-system process overhead and state management.

Figure 4.10 further represents an adapted process instance life cycle with minor activity updates. The representation in Reichert and Weber (2012)(pp.34) shows several input-output requirements for each process state. The OPM state objects for the internal process execution are extracted directly as; **created**, **running**, **suspended**, **completed**, **activated**, and **terminated**. The **activated** state is updated from the original **active** state. Additionally, the use of a gerund can interchange such that running represents a running process, or in process form **Process Running**. The process template creates a plausible set of process boundaries against which low level process instructions abstracts. A finite set of states such as these, coupled with the abstracted Android OS constants indicated in section 4.3 offer a variation matrix against which application level decisions for **Front-end System Set** establishes. The interactions for the simple process activity communication in Figure 4.10 is represented using a Design Structure Matrix (DSM).

The use of a DSM to represent the ISARC decision structure leans on the parameters extracted throughout this chapter and preceding chapters. The process mapping in Table 4.6 aligns to the structure contained in DSMWeb (2019) as opposed to Eppinger and Browning (2012)(pp.4). As a result of the inherent flexibility of n-squared matrices, a consistent view of decision outputs and decision inputs should exist in the ISARC decision approach to ensure comparable baselines for ongoing decisions relating to the ISARC. Therefore;

1. rows present the output to a cell in the appropriate column and
2. columns present the input received from a cell in the appropriate row.

Table 4.6: Process Activity State Transition DSM

Activity		A	B	C	D	E	F
Created	A		x				x
Running	B			x	x	x	x
Suspended	C		x				x
Completed	D						
Activated	E		x				x
Terminated	F						

Table 4.7: Process Activity State Transition DSM Clustered

Activity		B	C	E	A	D	F
Running	B		x	x		x	x
Suspended	C	x					x
Activated	E	x					x
Created	A	x					x
Completed	D						
Terminated	F						

Table 4.6 outlines a detailed structural interaction that represents the process transitions in the ISARC. Activity interactions for Row B Column C, and Row C Column B, indicate a bidirectional relationship between these process activities. Similarly, a bidirectional relationship exists between Row B Column E, and Row E Column B. Optimisation of the structural matrix is possible using a clustered input/output approach. The matrix reforms to appear as Rows:{BCEADF} and Columns:{BCEADF}. Table 4.7 presents a reorganised DSM for the aforementioned input/output clustering. The intersecting dead cells - black cells - will remain at the intersection of the row and column of the shifted column set. Clustering and reorganising is a known DSM handling approach, Eppinger and Browning (2012)(pp.6).

To validate the configuration for controller placement options discussed in section 4.4, an example of a control placement problem is noted in IoT, in Stankovic (2014). The decision parameters for feedback control placement of Agent (Human-in-loop) control positions at several places within a feedback control system. These are; *Outside of the Loop*, *Inside of the Controller*, *Inside the System Model* - namely within the system boundary, *Inside a transducer*, and *at various levels of hierarchical control*. These runtime placements play a vital role in the primary control purpose and those actions that require compensating behaviours. Given the configuration variations of the system parameters seen thus far, a set of structural model components such as a DSM provides the necessary coverage in reducing system design complexity.

4.4.1 Applied Process State Transition

In OPM state transition, section 4.4.1 indicated a set of transition entity attributes representing the state of a system at a point in time. These are represented as being either;

initial, default, or final (ISO/TC 184/SC 5 (E) (2014)). The definitions show that;

1. the *initial* state of an object shall be its state as the system begins operating, or its state upon generation by the system during operation. Therefore the initial state of the process transition is representative of the function and the objects state while the state is measured,
2. the *default* state of an object shall be the state in which the object is most likely to be upon random inspection at the time that the state is measured, and
3. the *final* state of an object shall be its state as the system completes operation or its state upon consumption by the system during operation, which is broadly the closing state measurement.

The entities in the set of states and transitioned states, represents a path through the system such as those seen in Table 4.8. Based on an analysis of the candidate **Front-end System Set** process requirements extracted from the OS request constant list, and coupled with the process state space, the requirements in Table 4.8 represent a series of behavioural aspects of a **Front-end System Set**. The transitions in the table represent the finite state transitions, represented as a process activity transition adapted from Reichert and Weber (2012)(pp.34). The interactions of the known process activity flow seen in Figure 4.10 - simplified in Table

Table 4.8: Proposed Request Interactions for ED/GW

Constant	Form	Function	Process State Transition
SIGINT	User/Process	Interrupt Requesting	running → terminated
SIGQUIT	User/Process	Termination Requesting	running → active → terminated
SIGABRT	Process	Abort Requesting	running → terminated
SIGIOT	Process	Abort Requesting	running → terminated
SIGKILL	Process	Termination Requesting	running → terminated
SIGUSR1	Process	User Defined Condition Requesting	running → active → terminated
SIGUSR2			
SIGTERM	Process	Termination Requesting	running → active → terminated
SIGCONT	OS/Process	Paused Process Continuation Requesting	suspended → running
SIGSTOP	OS/Process	Halt Process Requesting	running → suspended
SIGTSTP	Process	Stop Process Requesting	running → active → terminated
SIGBREAK	User/Process	Interrupt Requesting	running → terminated

4.6 - indicates that a baseline for architectural decisions is plausible for a subset of **Front-end System Set** decisions, and by extension other IoT-Ce decisions. The creation of pure gerund processes produce a consolidation of process transitions seen in Table 4.8.

Figure 4.11 introduces a self-standing process for **Front-end System Set** request handling. The system diagram proposes a testable series of interactions which uses the *SIGINT* request constant from Linux (2018) and integrated with the transitions from Reichert and

Weber (2012)(pp.34). System value drivers on the change of system state from **active** to **passive** propose a main operand in the SD. The modelling of a system interrupt has value in that rogue processes can be terminated on demand.

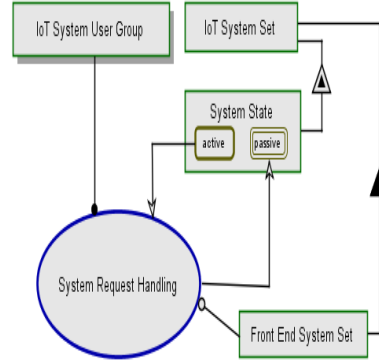


Figure 4.11: Function:ED System Request Handling SD

Listing 4.9: OPL:ED System Request Handling SD

```

1 IoT System Set exhibits System State.
2   System State can be active or passive.
3   active is initial.
4   passive is final.
5 IoT System Set consists of Front End System Set.
6 IoT System User Group is physical.
7 IoT System User Group handles System Request Handling.
8 System Request Handling requires Front End System Set.
9 System Request Handling changes System State from active to passive.

```

The SDs in Figure 4.11 and Figure 4.12 extends the IoT-Ce system value model modelled in OPCAT 4.2. The figures apply IoT-Ce specific requirements with inclusion of new **Front-end System Set** objects such as **Request** and **SIGINT** object state transitions. Figure 4.11 applies the repeatable model form and function set as the pre-installed ABS example in OPCAT4.2. The model however represents a smaller system in the IoT-Ce in that it only handles the interrupt function that a **Front-end System Set** experiences. The function is provides a direct attachment of the interrupt as required on the **Front-end System Set**. All *Things* represent a systemic view and thus all form part of the same boundary. Given the atomicity of the function, it could attach to any process, as previously argued in the case of the The **IoT System Control Handling** process. The SD1 for the **System Request Handling** processes appear in Figure 4.12.

Figure 4.12 represents a process execution that outlines a runtime testing approach. The solid red dots in Figure 4.12 traverse the interaction links between processes and objects and change the state of objects along the execution path. The OPL that represents this testable model also appears below. Model execution is a carefully planned process. Incorrect

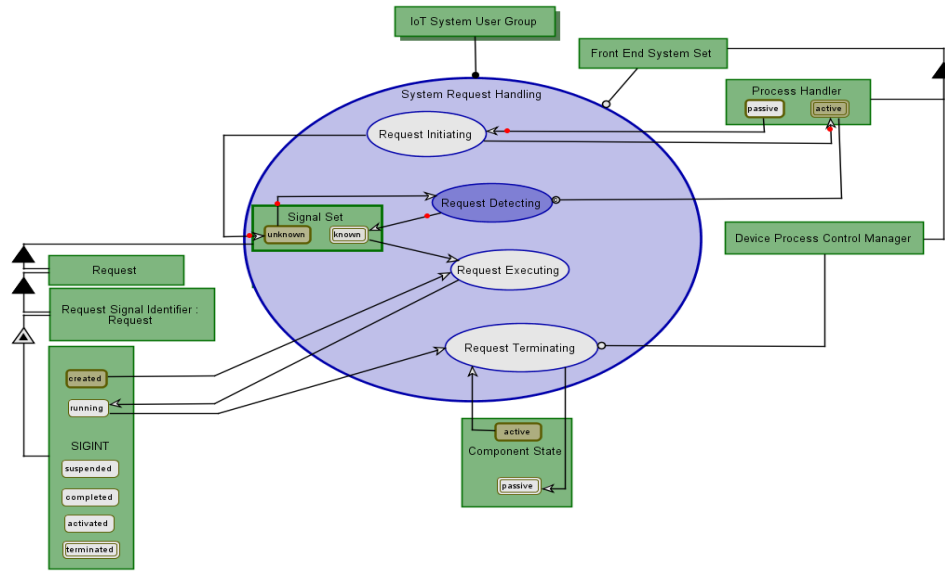


Figure 4.12: OPD:ED System Request Handling SDx

usage of links and objects will result in OPCAT errors such as *Agent Waiting* or *Activation Scheduling*, which causes a model to fail during OPCAT simulation.

Listing 4.10: OPL:ED System Request Handling SDx

```

1 Component State can be active or passive.
2   active is initial.
3   passive is final.
4 Front End System Set consists of Process Handler and Device Process Control Manager.
5   Process Handler can be passive or active.
6     passive is initial.
7     active is final.
8 IoT System User Group is physical.
9 IoT System User Group handles System Request Handling.
10 Request consists of Request Signal Identifier.
11   Request Signal Identifier exhibits SIGINT.
12     SIGINT can be created, running, suspended, completed, activated, or terminated.
13       created is initial.
14       terminated is final.
15 System Request Handling exhibits Signal Set.
16 System Request Handling consists of Request Initiating, Request Detecting, Request Executing, and Request Terminating.
17 System Request Handling requires Front End System Set.
18 System Request Handling zooms into Request Initiating, Request Detecting, Request Executing, and Request Terminating, as well as Signal Set.
19   Signal Set can be unknown or known.
20     unknown is initial.
21     known is final.
22   Signal Set consists of Request.
23   Request Initiating changes Process Handler from passive to active.
24   Request Initiating yields unknown Signal Set.
25   Request Detecting occurs if Process Handler is active.
26   Request Detecting changes Signal Set from unknown to known.
27   Request Executing changes SIGINT from created to running.
28   Request Executing consumes known Signal Set.
29   Request Terminating requires Device Process Control Manager.
30   Request Terminating changes Component State from active to passive.
31   Request Terminating consumes running SIGINT.

```

In summary, the proposed extraction of objects, processes, and states continues to provide a viable mechanism to stipulate individual requirements and decision parameters. As demonstrated in this section, the simplification of questions that filter the system form and system function can offer a means to maintain lineage in complex design. By traversing a natural hierarchy of system diagrams from the uppermost purpose level to lower-levelled signalling concerns in **Front-end System Sets**, the ISARC presents the possibility of linking high-level system requirements to lower-level IoT-Ce process states.

4.5 ISARC Form & Function Summary

This section summarises how the ISARC adheres to the primary systems architecture, form, and function questions outlined in Table 4.1. The summary aims to baseline time-independent requirements against which the functional design tests. The requisite processes that govern the initial state of objects for the system value delivering (SD1), data transmitting (SD1.1) can be found in Figures 4.2 and 4.3, respectively.

Maintaining clarity in the argument on using OPM to abstract a lower-level system process can challenge the boundaries of completeness and clarity indicated by Dori. Thus, the selected operating constants set the boundary for immediate process control in a **Front-end System Set** system boundary. The boundary of process control to a **Front-end System Set** is internal when the operating system control needs to manage device resource usage. The finite state space represented in Table 4.4 produces 36 possible interaction points within the Android **Front-end System Set** operating system. Additionally, a series of process variations that create a constrained runtime process sequence baseline appears in Figure 4.10. These inputs create a finite system boundary that claims to model the abstracted IoT-Ce's static constructs and constraints.

As indicated in ISO/TC 184/SC 5 (E) (2014)(pp.13), an OPM object can exist without a process. However, a process can not exist without an object as this process is the fundamental transformation entity in the system. At a minimum, Chapter 4 reinforces the hypothesis limitation for the boundary considerations of the architecture. This consideration eventually governs the integrated system architecture boundaries en route to the validated system model evidence. This summary intends to produce the essential alignment of the thesis to the systems architecture approach instead of arguing the merits of why form exists before function and before systems architecture. The overarching proof of object arrangement as a configuration indicates that the runtime arrangement as the basis of state transition is fit for OPM given the object dependability principle in Dori (2002)(pp.61). A systems belief that

drives the ISARC is that it positions in an existing multi-levelled hierarchy. The hierarchy of frameworks, to clocks, towards self-regulation is a general system hierarchy noted in Dori (2002)(pp.386) and Skyttner (1996)(pp.71).

- **3.1::System Architecture::Main Object Process Transformation** : The main object and process transformation is outlined in Figure 4.2. This primary process exists at one level lower than the system purpose. The delivery of the system value; [IoT Edge System Value Delivering](#), is represented as processes for two of the three IoT domains of interest; [IoT Data Generating and Acquiring](#) and [IoT Data Transmitting](#).
- **3.2::System Architecture::Additional Object Process Transformations**: Additional object process transformations such as [IoT Data Analysing](#) is a non-ideality shown alongside the main process transformation in SD1.
- **3.3::System Architecture::Supporting Object Process Transformation**: Supporting objects are represented via environmental objects such as the [Control Handling](#) process connected to the [IoT Data transmitting](#) process in Figure 4.4.
- **3.4::System Architecture::System Boundaries and Interfaces** : System boundaries are represented by shared processes. Boundary typed process interactions appear in Figure 3.9. Boundary processes are modelled in the environmental control detail in Figure 4.6. The process that indicates the primary boundary is the [Failed State Notifying](#) process embedded in the core control problem. This control constraint appears in the control problem adapted from Janert (2013). Additionally, the [Failed State Notifying](#) process is adapted from the notify constants extracted from the request constant library. The process that consumes and changes the [request handled](#) object from [unhandled](#) to [handled](#), is also attached to an unhandled state with a condition link.
- **3.5::System Architecture::Process Execution and Priority** : Process execution priority appears as sequential flows via top-down execution of processes indicated in Figure 4.4. The process outputs for each zoomed-in process yield a single object consumed by the next process in the zoomed process boundary.
- **3.6::System Architecture::Parallel Process Execution and Priority** and **3.7::System Architecture::Timing and Constraints**: The model alignment requirements for these systems architecture characteristics are explored in the next chapter on the analysis of [Device Identification](#), [Device Discovery](#) and [Device Management](#).
- **1.1::Form::Abstracted System Representation**: The delivery of IoT System value is largely abstracted from the final SDs that model the change of process state

within signalling constants. The representation of the interrupt constant in Figure 4.11 provides a concrete proposal of the change in state of a created process to one that is running. A subsequent implementation model of interrupt behaviour will solidify this viewpoint because a running process can terminate in a **Front-end System Set**.

- **1.2::Form::Decomposable Form Elements:** form structures that can decompose, use the notion of a **Front-end System Set**, in relation to a more coarse-grained IoT System Set. This approach models the exact hierarchy indicated in current edge systems research. Although current IoT edge research does not structure in ISO OPM-19450 terms, the resultant ISARC positions the need for the existence of physical and informatical objects before process execution can occur. This pre-existence is a view held in ISO OPM-19450.
- **1.3::Form::Spatial or Connectivity Structures:** Connectivity structures formalize using instrument and event links that enable execution of the lower-level processes such as **System Request Handling**. Spatial considerations are not yet present as the internal system boundary concerns only regard inbound requests and handling of events within a time-bound constraint. This time constraint is essential when validating the parameter outputs against the *failure trace data set* in Chapter 5.
- **1.4::Form::Main and Supporting Systems Composition:** The composition of the primary system is represented in the system purpose SD. Interactions between the **IoT System User Group** and the **IoT System Set** appear as a management relationship, with both agent and instrument links.
- **1.5::Form::System Boundaries and Interfaces:** Primary system boundaries are present in reference to possible environmental systems that could impact the primary system process. An external control handler appears in Figure 4.5 such that a condition link enforces an occurs-if relationship in the execution order of the process. For event link constraints, ISO/TC 184/SC 5 (E) (2014)(pp.90) indicates that event links should not cross the system boundary as this could interfere with the execution order of a synchronous process.
- **1.6::Form::Use Context:** The primary use context is the value delivered via an increase in dependability. The testable model set outlines the scenario where a process is interrupted abruptly by a user. This scenario is one of the highest system unavailability forms - where a system resource is unreachable. The scenario does not indicate a failed system state from where recovery is impossible. The terminated process merely indicates that a specific service is not available at a point in time. This

state is comparable to a user-induced random failure at some point in a measurement period.

- **2.1::Function::Primary Externally-delivered Value:** The primary delivered value is increasing system dependability using the IoT-Ce system set. In the case of the front end and near end system set, the dependability of these systems is the overarching *ility under which reliability is a supporting metric.
- **2.2::Function::First and Second level Internal Objects, Processes and States:** An internal system diagram is presented in Figure 4.3 and then unfolded as an object state change diagram with the next level of detail in Figure 4.6
- **2.3::Function::Functional Architecture through Internal Process Interactions:** A function to form mapping is handled within several SDs that show object transformation by consumption and result links. Examples of the these can be found in [IoT Edge System Value Delivering](#) which changes [dependability](#) from [low](#) to [high](#), and [Device Identifying](#) which changes [identification](#) from [unknown](#) to [known](#).
- **2.4::Function::Secondary Externally-delivered Value:** A secondary external value delivery can be claimed due to increasing dependability, in that a more efficient data delivery channel and analysis can occur. Although the primary functions of transmitting and sensing are modelled in the SDs, the closed-loop analysis of system value delivery might be needed to achieve a higher system beneficiary value.

4.6 System Configuration Decision Model

The systems architecture for an IoT-Ce uses OPM to model the hierarchical system's runtime use context. A substantial set of OPM-extracted parameters produce the support for an expanded set of ongoing parameter-based architecture decisions against this IoT-Ce hierarchy - for future work. Additional IoT and IoT-Ce decision parameters are presented to support ongoing parameter extraction for areas other than the IoT-Ce. These parameters appear in Appendix B. An architecture decision principle formation in Crawley et al. (2016)(pp.326-327) shows three heuristics for creating architecture decisions. These heuristics are; *a bounded architectural decision space*, *decisions that impact metrics significantly*, and *decisions that relate to an architecture specifically, and not to design decisions*. The following decision model outlines the integration of the essential IoT-Ce parameter sets from Chapters 3 and 4. The decision model;

1. **Model 1** - is known to contain IoT-Ce subsystem interactions, as indicated in the aggregated architecture options overview presented in Figure 2.1, and extracted into ISARC parameter Table 3.6,
2. **Model 2** - are perceived component interactions built from IoT definition extracts into Table 3.4 and 3.5. These can be seen in the main SD hierarchies in Figures 4.1, 4.2 and 4.3,
3. **Model 3** - are known component process states and constants extracted into summarised form into Table 3.7, but expanded in Tables 4.4 and 4.5. These are also indicated in Figures 4.8 and 4.9, and
4. **Model 4** - are known component process transitions extracted into Table 3.8, and indicated in Figure 4.10.

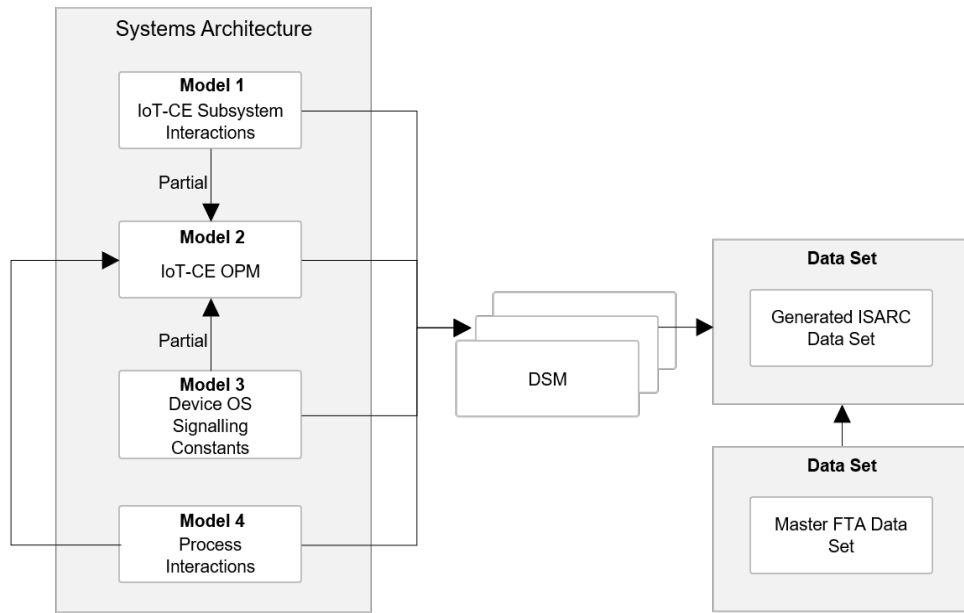


Figure 4.13: System Configuration Decision Model

Figure 4.13 outlines the model inputs described in preceding chapters as a collective body of work. This body of work represents a decomposable value-to-process interaction lineage for the IoT-Ce using a pure systems architecture approach. Figure 4.13 presents the four model types that support the system boundary. As noted in the model summary above, a limited configuration space contains the parameter set for each model type. The extraction of specific parameters aligned to the IoT domains creates a multi-layered hierarchy that binds parameters into each layer. This multi-layered approach has resulted in using a **dependability** parameter in the SD node layer of the OPM's top-level OPD. This parameter relates to the

Table 4.9: Decision Model 1 for IoT-Ce Domain Interactions for Figure 2.1

Subsystem/Component		C1.1	C1.2	C1.3	C1.4	C2.1	C2.2	C2.3	C2.4	C2.5
Communication	M1.C1.1		x		x		x			
Hardware Abstraction	M1.C1.2	x		x	x					
Operating System	M1.C1.3		x							
Remote Management	M1.C1.4	x	x				x			
Data Management and Messaging	M1.C2.1						x			x
Communication and Connectivity	M1.C2.2				x	x		x		x
Application Runtime	M1.C2.3						x		x	x
Operating System	M1.C2.4							x		o
Remote Management	M1.C2.5					x	x	x	o	

primary beneficiary value delivery for the proposed systems architecture - [IoT Edge System Value Delivering](#). The zooming and refinement between model nodes SD, SD1, and SD1.1 provide a hierarchy that mimics the IoT domains of interest. An abstract view of the state of value delivering and dependability (SD) is attached to the IoT core domains of interest (SD1). These SDs are further attached to a specific function of transmitting data using an [Front-end System Set](#) (SD1.1). The final characteristic relating to the applicability of OPM as the elected representation of the IoT-Ce systems architecture is the representation of the runtime contexts of processes in an abstracted [Front-end System Set](#). The lineage created in the system diagrams proposes a single contribution; however, a larger interaction DSM proves the flexibility of applying OPM for the ISARC on an ongoing basis. The remainder of this section proposes the consolidation of parameters to structure a series of components and process DSMs. This section produces the final design matrices, which are coded into Python and simulated for approximation against the DNS Failure Trace Data sets discussed by Pang et al. (2004), Kondo et al. (2010), and Epema et al. (2014).

4.6.1 Decision Model 1 Inputs

Decision model 1 creates a static baseline based on the aggregated architecture options overview is presented in Figure 2.1. Within each of these viewpoints, architectural options can be analysed simply by using a novel structural matrix. For example, a DSM to outline component interfacing appears in Table 4.9.

Rows C1.1 to C1.4 contain system capabilities on a constrained device ([Front-end System Set](#)). The C1.x subsystem cluster contains an [OperatingSystem](#) that interacts internally toward the [HardwareAbstraction](#) represented by C1.3 $\rightarrow$ C1.2. The interactions in the C1.x subsystem indicate a tightly bound set of interactions where all components interact. The core components interact via the [HardwareAbstraction](#) layer, essentially creating a single point of access from the [OperatingSystem](#).

Table 4.10: DM2:A Candidate DSM for IoT-Ce Interactions for the ISARC SD through SD1.1

Location	Type	Entity		M2.1	M2.2	M2.3	M2.4	...
SD, SD1	Process	IoT Edge System Value Delivering	M2.1					
SD, SD1	Object	IoT System User Group	M2.2					
SD	Object	IoT System Set	M2.3					
SD, SD1	Object	Near End System Set	M2.4					
SD	Object	NE System Dependability	M2.5					
SD, SD1, SD1.1	Object	Front End System Set	M2.6					
SD, SD1	Object	FE System Dependability	M2.7					
SD, SD1	Object	IoT System Controller	M2.8					
SD1	Process	IoT Data Generating and Acquiring	M2.9					
SD1	Object	Sensed Data	M2.10					
SD1	Process	IoT Data Transmitting	M2.11					
SD1	Object	Transmission Attempt	M2.12					
SD1	Process	IoT Data Analysing	M2.13					
SD1	Process	IoT System Control Handling	M2.14					
SD1	Object	Edge Device	M2.15					
SD1	Object	Sensing Capability	M2.16					
SD1	Object	Communication Capability	M2.17					
SD1	Object	Configuration Set	M2.18					
SD1	Object	System State Data Set	M2.19					
SD1.1	Process	Device Identifying	M2.20					
SD1.1	Object	Identification	M2.21					
SD1.1	Object	Assigned Address	M2.22					
SD1.1	Process	Device Connection Establishing	M2.23					
SD1.1	Object	Device Connection State	M2.24					
SD1.1	Process	Device Data Sending	M2.25					
SD1.1	Process	Connecting Terminating	M2.26					

4.6.2 Decision Model 2 Inputs

The second decision model extracts the core ISARC contribution. The ISARC diagrams contain object, process, and state components. The modelling of OPM-based DSMs are already proposed for process-based architectures, Sharon et al. (2012). The second contribution of the study - a simulated ISARC model proof - is not based on transforming the OPD elements into a DSM efficiently. Recently, Sobol and Dori (2020) already cover OPM hierarchy flattening using DSM. Rather, the DSM offers a compact structure for ISARC architecture decision placement. Table 4.10 produces a DSM based on Figures 4.1 (SD: System Purpose), 4.2 (SD1: Primary Function) and 4.3 (SD1.1: In-Zoomed Primary Process). The core focus in this ISARC decision DSM is the representation of known OPM links included in the source figures.

Table 4.10 shows the strength of a single compact DSM sourced from 3 different OPM SD levels. A redacted DSM is presented to outline the format however greater focus is placed on the codification of this DSM into Python, in Chapter 5.

Table 4.11: Decision Model 3 for IoT-Ce ED/GW Operating System Signalling

Type	Constant		M3.1	M3.2	M3.3	M3.4	...
Notify	SIGHUP	M3.1					
Request	SIGINT	M3.2					
Request	SIGQUIT	M3.3					
Notify	SIGILL	M3.4					
Notify	SIGTRAP	M3.5					
Request	SIGABRT	M3.6					
Request	SIGIOT	M3.7					
Notify	SIGBUS	M3.8					
Notify	SIGFPE	M3.9					
Request	SIGKILL	M3.10					
Request	SIGUSR1 SIGUSR2	M3.11					
Notify	SIGSEGV	M3.12					
Notify	SIGPIPE	M3.13					
Notify	SIGALRM	M3.14					
Request	SIGTERM	M3.15					
Notify	SIGCHLD	M3.16					
Notify	SIGSTKFLT	M3.17					
Request	SIGCONT	M3.18					
Request	SIGSTOP	M3.19					
Request	SIGTSTP	M3.20					
Request	SIGBREAK	M3.21					
Notify	SIGTTIN	M3.22					
Notify	SIGTTOU	M3.23					
Notify	SIGURG	M3.24					
Notify	SIGXCPU	M3.25					
Notify	SIGXFSZ	M3.26					
Notify	SIGVTALRM	M3.27					
Notify	SIGPROF	M3.28					
Notify	SIGWINCH	M3.29					
Notify	SIGIO	M3.30					
Notify	SIGPOLL	M3.31					
Notify	SIGLOST	M3.32					
Notify	SIGPWR	M3.33					
Notify	SIGINFO	M3.34					
Notify	SIGSYS	M3.35					
Notify	SIGUNUSED	M3.36					

4.6.3 Decision Model 3 Inputs

Decision model 3 represents an interaction matrix for both request and notify signal sets in the NodeJS OS library. Although the primary interrupt use case is a request signal, a larger coverage interaction matrix allows for a larger populated data sample that aims to be generalised in Chapter 5. Additionally, the baseline process interactions outlined in Table 4.6 coupled with a larger number of completed signal end states, enables the benefits of having a finite process transition state space without unduly extrapolating interactions. This approach supports a well-defined system boundary. Table 4.11 represents a redacted 36 * 36 element matrix. The ability of a process state to reach another governs the **Front-end System Set** component interaction space. The primary optimisation is to achieve the desired state while achieving efficiencies in reduced process switching and inefficient state management. The allowable runtime component interactions validate in Chapter 5.

Table 4.12: Decision Model 4 for IoT-Ce Process Interactions

Activity		M4.1	M4.2	M4.3	M4.4	M4.5	M4.6
Created	M4.1-A		x				x
Running	M4.2-B			x	x	x	x
Suspended	M4.3-C		x				x
Completed	M4.4-D						
Activated	M4.5-E		x				x
Terminated	M4.6-F						

4.6.4 Decision Model 4 Inputs

Decision model 4 outlines design decisions that define a fixed set of baseline process interactions. An example of the primary baseline configuration can be seen in section 4.4, with an early view of the DSM presented in Table 4.6. The static representation of state:1 (functioning) interactions is indicated in Table 4.12. The interactions set the baseline constraints for the known interactions that could occur in bound-process execution. In Reichert and Weber (2012)(ch.5), flexibility in process control presents as fitting multiple behavioural contexts. Reichert and Weber (2012) sees variations in process models derived from a base model as generally error-prone. Thus, the execution order of the known process activities serves as a configuration baseline against which variations test, but ongoing variation will result in multiple execution paths. In state transition, a set of transition entities represent the known reachable state of a system. These entities in the set of states and transitioned states represent a path through the system. The allowable process state transitions from the DSM in Table 4.12, shows the following possible transitions;

1. Created => Running or Terminated
2. Running => Suspended or Completed or Activated or Terminated
3. Suspended => Running or Terminated
4. Activated => Running or Terminated

The associated state space for each scenario is thus;

1. $S_a = \{B;F\}$
2. $S_b = \{C;D;E;F\}$
3. $S_c = \{B;F\}$
4. $S_d = \{\}$
5. $S_e = \{B;F\}$

6. $S_f = \{\}$

In Table 4.12, the configuration of a running process activity indicates that it could have been bypassed entirely by the creating process activity. It is inferred that this represents a failed process creation. The parameter space for this process could therefore be $S_a = \{F\}$. $S_d = \{\}$ and $S_f = \{\}$ represent empty sets which are final states in the process transition.

4.7 Chapter Summary

Chapter 4 produced the evidence for considering OPM for the IoT-Ce. Using an extracted parameter set, the ISARC shows that parameter-based approaches can handle heterogeneous structures and functions. This parameter-based extract can occur across IoT literature, frameworks, and implementation architectures. A standardised usage of the OPM symbols produces a well-structured set of diagrams (OPDs) and the required automated text generation form (OPLs). By applying a simplified systems discovery method, creating objects and processes that fall into either systemic and environmental boundaries is achievable. In the argument that a broader IoT body of knowledge is available over different time horizons, the recommendation to use OPM's fundamental principles outweighs the modelling of a systems architecture in competitive systems approaches. OPM requires a core understanding of the operating structures of systems. In contrast, competing methods and languages focus on cross diagram representation. The use of a single naturalised symbol set in OPM has enabled the specification of value delivery, attached to primary IoT domains, and then finally towards a physical specification of an abstracted **Front-end System Set** while keeping principles such as hierarchy, boundary, and state in mind.

Chapter 4 further produces the first cycle of system identification as a step into uncovering additional primary and additional supporting functions that could form part of the primary system value SD. The flexibility of the OPM assures that active learning advances while modelling the systems architecture. The chapter finalizes the consolidation of the parameter extracts that form the basis of model proof in chapter 5. Chapter 4 provided an optimised requirements model for extracting objects and process parameters that serves as a precondition for the runnable systems model. Chapter 5 consumes the ISARC parameter extracts achieved in Chapters 3 and 4 towards a simulation that should drive increases in the fidelity of ongoing architectural decisions for the IoT Systems Architecture for Connected-Edge Runtime Configuration.

The outline of Chapter 4 has driven five core systems architecture requirements;

1. conformance to OPM symbol usage for the structural and behavioural aspects of IoT-Ce systems that drive configuration variability,
2. the main system purpose and primary system interactions,
3. supporting objects and process transformations,
4. system interfacing and interactions that support control model types, and
5. process variation and configuration,

In line with the primary and secondary contributions of the thesis, Chapter 4 finalises the proposed standards-based systems architecture that outlines the runtime configuration of IoT application components found in the IoT-Ce. Chapter 5 aims to survey the form and function of the underlying architecture to provide an understanding of decisions that guide reliability characteristics that support state-awareness - aligned explicitly to a DNS faulted data set.

Chapter 5

Examples and Evidence: ISARC Model Validations

Chapter 5 provides an example of extensions to the model framework presented in Chapter 4. These extensions apply the evidence of using ISO OPM-19450 to model systems hierarchies for the IoT-Ce. The application of an executable model indicates that IoT system value to physical device mapping is possible using a single notation optimised for graphic and textual representation. On model evidence, earlier SDs expand to a position that enables model testing in OPCAT 4.2.

Chapter 5 also solidifies the views on architectural decision variation. Preceding chapters have discussed vertical themes - System Architecture, IoT-Ce, and Configuration - which culminated in creating a definition-to-OPM filter A. A parameter extraction process based on principles by Dori (2002), ISO/TC 184/SC 5 (E) (2014) and Crawley et al. (2016). The IoT requirements extraction process presents a compact set of models that contain singular objects that compose larger objects, with the required system hierarchy placement. A Design Structure Matrix (DSM) presents a compact and representative view of the decision model against which the evidence validates.

The proposed ISARC architectural structure for continuous decisions appears in Chapter 5. An applied Process DSM represents the proposed ISARC decision model. A combination of Process and Product DSMs apply instead of Organisation DSMs for the decision model. The interest in ISARC architecture decisions lies in decisioning based on the continuous flow of decisions that impact runtime configurations for the IoT-Ce. Thus, interactions within an organisational structure are not needed. The representation of the ISARC, however, uses both Product and Process DSMs. As indicated in the system configuration DSMs in Chapter 4, the parameter content argues for static architectures that support runtime behaviours.

Namely, both structural and runtime architectures. As per previous summaries, function (state change of an object) executes on form (structure) to produce system value. Further considerations apply to control decisions in the form of a PI Controller. As seen in earlier discussions on autonomous systems design, the nature of feedback impacts system configuration. Although the PI offers standard mechanisms against which proportional control for immediate error correction and accumulated error testing, the decision model data set includes random connectivity time and failure state time as standard data set characteristics.

Chapter 5 produces an exciting look into the filtered parameter set and its architecture decision-data generation possibilities in Python. The output of the simulations test against a historical data set - the DNS Failure Trace Archive produced by Pang et al. (2004).

5.1 Primary SD, SDx, SDx.x Model Overview

The primary systems model presents parameters that support increases in dependability for mobile IoT users. Therefore, the systems model presents a hierarchy of model diagrams that argue for the placement of components at the correct level of the IoT-Ce system's hierarchy. The structural view of the model presents time-independent system objects to support functional design. The topmost level SD node presents the purpose of the system that the model aims to communicate. The SD1 provides the system's capabilities, with SD1.1 providing a more detailed view of the primary processes that aggregate upwards towards the purpose of the system. Figure 5.1 represents the model hierarchy. The primary model overview delivers

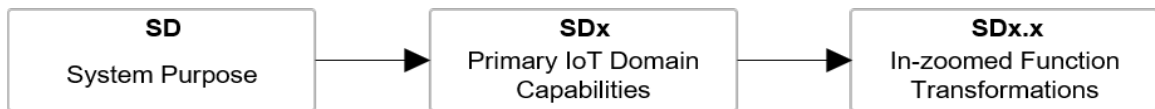


Figure 5.1: System Model Overview

three applied SD sub-hierarchy use cases. A structured system design viewpoint represents the implementable system, as seen from a system engineer's perspective. It is the IoT-Ce systems viewpoint that will be deployed and managed. Crawley et al. (2016)(pp.67) defines the form as the "concrete" realisation of a system, or more aptly, the structure of a system that exists for some time. The period represents current and future requirements for the deployed system. The form of the system instruments the function and produces an emerged value of the system. Applied SD sub-hierarchies are those SDs that appear at lower levels to the central system diagram geared towards specific problem resolutions. Owing to the incredible flexibility of the OPM ISO-19450, the hierarchy's depth is not hardbound; however, it requires a practical balance between completeness and clarity of the model content.

This balance is a known characteristic of OPM Dori (2011). Similarly, this balance applies to the system decomposition found in Figures 4.1 through 4.6.

These figures show that the primary beneficiary value of the IoT-Ce system (Constrained Device, Gateway, Smartphone, Front-end, and Near-end System Set) is to increase dependability in how generation and acquisition, and transmission occurs. The analysis of IoT data in the IoT-Ce represents an additional object process transformation that ultimately supports the IoT data generating and acquiring, and transmitting processes. Thus, the first system boundary discussed is a shared process that enables subsystem interaction. In its earlier representations, SD represents the system value, SD1 is an in zoomed process, and SD1.1 further refines an in-zoomed process [IoT Data Transmitting](#). The boundary condition set in the preceding chapter sets an environmental control configuration. The primary process of handling system requests related to [IoT Data Transmitting](#), creates an external invocation to the [IoT System Control Handling](#) process.

Within the sub-hierarchy, a component set is the system set required to execute the system's main processes. This enabler - the instrument - is a [Front-end System Set](#) with capabilities that enable device process handling, network connectivity, or capabilities that manage components that sense the physical environment. These interface forms appear in earlier chapters on accessing mobile and device operating system states. However, considering the edge device as a single value providing thing, one might under specify or misrepresent the system value as abstracted to an application running on the [Front-end System Set](#) or value attained from accessing a specific sensor reading. To counter this misrepresentation, the systems architecture maps the structural components to the mechanisms that apply a change to the structure. A baseline structural overview appears as a layered framework containing levels of abstraction away from core hardware requirements such as concurrency and scheduling. A consolidated framework representing a convergence of a standard layered Android stack and a sensor node structure from a sensor architecture appears in Figure 5.2. This convergence extends the understanding that other abstracted functions such as communication, processing, and memory connect to the hardware drivers and exposed northbound towards IoT-Ce client applications. Examples of these are already present in the `os.networkInterfaces()`, `os.freemem()`, and `os.totalmem()` methods indicated in Linux (2018). Finally, the sub hierarchies provide detailed views on reliability characteristics by attaching reliability as a key systemic dependability driver. Reliability manifests as an aggregation or specialization object depending on the inheritance characteristics intended by the model. A systemic reliability driver manifests as a specialized dependability driver that is controllable

and exists within the systems boundary. Hierarchy nodes that live under the main hierarchy create a boundary that protects cascading system faults and failure. A view on hierarchical failure classifications in Høyland and Rausand (2009)(pp.86) sees failures classed as; Intermittent, Extended, Complete, Partial, Sudden, and Gradual. A systemic reliability driver thus exists as a controllable parameter that manages systemic failures. Figure 5.2 represents the consolidation of the layers within the **Front-end System Set** subsystem boundary. The **Front End System Set** is a systemic object that proposes to drive dependability increases in the main SD. The **Front End System Set** interaction with a primary processes appears in Figures 4.3 and 4.11

In their view on the architectural assumptions in a sensor network, Bera et al. (2016)

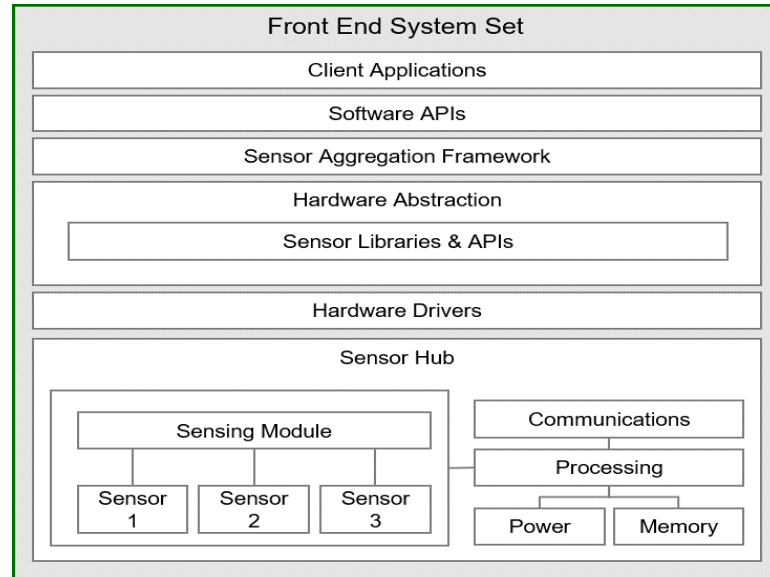


Figure 5.2: An Unfolded Front-end System Set. An integration between Android (2017) and Bera et al. (2016)

proposes a sensor connection based on a uniform randomly placed node. This behaviour effectively means that an **Front-end System Set** can move from position 1 to position n randomly. Essentially, supporting this node placement is two system instruments. One which manages ED/GWs and a second that states the configuration of the sensor network. In the ED/GW case, it is a self-contained component. However, topology and configuration knowledge is a specific function that could appear as either systemic or environmental. For brevity, the parameter **Knowledge Handling** references the stated parameter in Table 3.9. This split indicates that the Bera et al. (2016) proposed architecture, which splits application, control, and infrastructure aligns with a known split such as those proposed in the

converged abstracted framework, which splits configuration type, device integration, and control. This split is seen in Figure 2.17.

5.2 SDx.x.x Model Extensions

The sensor hub layer in Figure 5.2, informs the ISARC concrete node structure in the systems architecture. The continual smoothing of requirements leverages the OPM active-learning principle. Systems questions relating to systemic parameter treatment also includes additional parameters for communication, processing, and storage. This treatment considerably increases the variability of the operating modes of the sensor hub. The analysis of the operands (object transformations) expands the possible list of parameters that could impact architectural decisions for **Front-end System Set** reliability. For **Communication & Connectivity** to occur (M1.C1.1 and M1.C2.2), it is proposed that a communication capability exists and that **Data Transmitting** requires an **active communication capability**. Regarding the IoT System Primary Process OPL in 4.1.2 it shows that;

1. **device identification** is a candidate systemic or environmental parameter,
2. **device discovery** is a candidate systemic or environmental parameter, and
3. **device management** is a candidate systemic or environmental parameter.

The main system diagrams expands to contain processes that support the primary processes of **Identifying**, **Discovering** and **Managing**. As indicated in the earlier Figure 4.3, **Device Identifying** is the first executable process in the primary process **IoT Data Transmitting**. In the Figure 4.3 OPL line 32, **Device Identifying** yields an **Assigned Address**. Thus, an in-zoomed process for **Device Identifying** will show that **Device Addressing** is a sub process of **Device Identifying**. Based on link migration requirements in OPM, the instrument link attached to the **_address** parameter in Figure 5.3 can be moved from the SD1.x to the parent SD1 in Figure 4.3. This link migration is the responsibility of the modeller as stipulated in ISO/TC 184/SC 5 (E) (2014)(pp.96).

Each behavioural viewpoint in the sections that follow contains an OPD, OPL, and DSM.

5.2.1 Device Identifying

For identification, the re-purpose of existing standards to meet the needs of IoT use cases is proposed as a state-of-the-art approach to aid in component reuse and increased system interoperability. Examples of reuse manifest in the reuse of Domain Name Systems to support object naming services for edge-device identification, as seen in EPCGlobal Inc. (2008).

An **Address** assignment-focused data set for a two-levelled hierarchy appears in Pang et al. (2004). Pang et al. (2004) positions local (in-system-boundary) versus authoritative (out-of-system-boundary) DNS server subsystems. In the DNS address assignment flow, fewer highly loaded servers exceed the count of variable sparsely deployed local name servers. For clarity, local name servers apply in-boundary thinking as an IoT Gateway for address resolution.

The identification of a device per the DNS standard contains three core components, Pang et al. (2004). Client resolvers query local domain name servers that in turn query authoritative server components. The three subsystems uniquely form a single domain name system.

Addressing

Figure 5.3 represents the Device Identifying function containing the **Device Addressing** process using limited *Things* to produce a simplified SD1. The associated OPL for the objects and process in Figure 5.3 is outlined below.

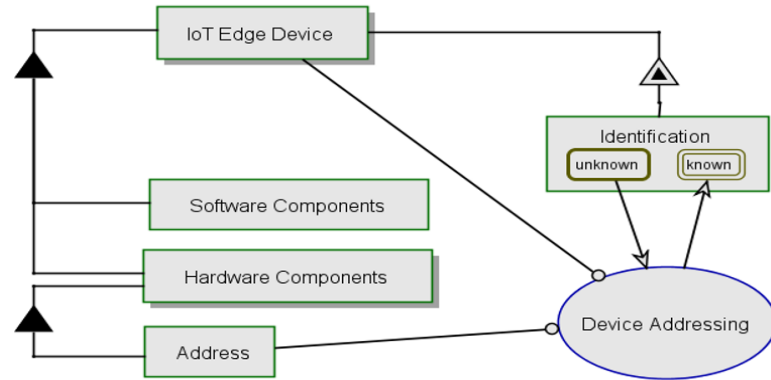


Figure 5.3: Function:OPD Device Identifying

Listing 5.1: OPL:Device Identifying Function

```

1 IoT Edge Device is physical.
2 IoT Edge Device exhibits Identification.
3   Identification can be unknown or known.
4     unknown is initial.
5     known is final.
6 IoT Edge Device consists of Software Components and Hardware Components.
7   Hardware Components is physical.
8   Hardware Components consists of Address.
9 Device Addressing requires Address and IoT Edge Device.
10 Device Addressing changes Identification from unknown to known.

```

The device addressing DSM in Table 5.1 has input and output that is declarative. The DSM clarifies the requirements for a possible set of parameters in an addressing decision. The DSM indicates that both software and hardware are interconnected by denoting an *r* (required) in Row B Column C and Row C Column B, but that the hardware component has an optional

Table 5.1: DSM:Device Identifying Decision

Configuration Parameter	A	B	C	D	E	F
IoT ED	A				r	r
Software Components	B		r	o		
Hardware Components	C	r		r		
Address	D	o				o
Unknown Identification	E					
Known Identification	F			o		

Address relationship. This might be true when **Front-end System Set** instances do not have direct internet connectivity, namely, they exist without IP connection capability. The **Address** referred to in the DSM appears on a hardware component such as a physical media access control (MAC) address. One could argue that the MAC address is also an abstraction of the physical component or merely a representation of a serial number that identifies hardware manufacturer and device. However, the abstraction occurs upwards towards the application layer instead of downwards into the physical device or materials used to produce components in the physical device. The modelling of lower-level material relationships is not too complex to imagine using OPM, using principles of hierarchy, boundary, and state.

5.2.2 Device Discovering

For device enrolment to occur into the IoT-Ce, certain physical constraints exists. A device has to exist to be discoverable before an enrolment can occur. Device discovery can exhibit three possible behaviours - fully automated, partially automated, and manual. As seen in section 2.2.2.2 seamless connectivity has several systemic requirements such as the existence of a **device**, **router** and **resource**. In these requirements, specialization of device roles such as **host** and **sensors** can exist. In addition to a more general process such as **Resource Addressing** or **Device Addressing**, **Resource Routing** and **Resource Connecting**, parameter extraction for **Front-end System Set** discovery protocols using definition-to-OPM extraction can thus be extrapolated from IoT protocols in section 2.2.2.2.

To enrol a device into the IoT-Ce, decisions relating to data persistence handling need to validate against the architecture. These architecture decisions are necessary as the **Front-end System Set** could advertise new or existing data in the IoT-Ce system registry. Two possible decisions are; cached data or persisted data. Cached data can be processed in memory and then persisted to increase the reliability of data exchange. In the event of a cache failure, a backup of the data retrieves from the persisted data store. Data that is elected to persist without primary caching is sent directly to a data store. Some forms of asynchronous commit happen on the data store to persist the data. There could, however, be some intermittent caching to aid in the commit process. However, this caching aids in

short term persistence - an additional or supporting function - and not as a primary function. This persistence behaviour is a known pattern in large-scale mobile telecommunication architectures. In the case of a mobile device and subscriber identity module (SIM) card, data lives at several architectural nodes such as the Equipment Identity Register (EIR), Home Location Register (HLR), and other service authentication nodes. Once the device is powered and attached to a mobile network, signalling flows connect and authenticate the device and SIM combination to enable mobile services such as voice or data traffic usage. The **Front-end System Set** enrolment pattern is seen to mirror this behaviour. A sensor hub will need permission - either granted automatically or approved from an IoT gateway waiting queue - to use the communication network to transmit sensor measurements via the **Front-end System Set**. Traversing the **Front-end System Set** requires the handling of data exchange between the layers indicated in Figure 5.2. An alternative is that the **Front-end System Set** could correlate the nearest active neighbour to the device and instruct direct communication with an asynchronous publish to ensure data consistency between nodes.

The seamless enrolment of devices is a key requirement for low cost, low power, or lossy devices in an IoT-Ce ecosystem. This requirement is primarily due to the device population size and variation that exists in IoT-Ce devices. The architecture decisions that allow for variations of the three enrolment options produce different scales of pre-existence requirements. For example, fully automated enrolment requires that devices contain some levels of intelligence on dealing with sleep and standby modes. These parameters can either be environmental, where less concern holds on the control of these modes. Alternatively, the control of these modes can be systemic, which places direct interest on the impact of decisions relating to different classes of sleep mode functions. For example, should a **Device Enrolment Notifying** process trigger a **Connection Terminating** process after **Device Enrolment Completing**, and then execute a **Device Sleep Mode Initiating** process? Logically, yes, given a strict sequence requirement, however, OPM also supports parallel process execution, which forms part of the architecture-form-function approach indicated in Table 4.1 (3.6 System Architecture Parallel Process Execution and Priority)

The system beneficiary in enrolment processes could be the **Front-end System Set** itself, having the problem occurrence stated as; **Changing Device Battery Usage** from **high** to **low**. The primary process will change an attribute such as **Sleep Mode** state from **active** to **passive**. Thus, the *Problem Occurrence, Purpose, Primary Function, Enabler, and Environment* approach applies to specific problems such as protocol selection for IoT, as seamless enrolment capabilities might be constrained or supported by the **IoT Protocol Set**.

Front-end System Set joining use cases executes as one of three object transformation types. The Architecture Reference Model (ARM) for IoT, seen in Bassi et al. (2013), outlines these use cases as;

1. Automated Device Discovery and Enrolment,
2. Automated Device Discovery and Partially Automated Enrolment, and
3. Manual Device Discovery and Enrolment.

On randomness, and as indicated by Bera et al. (2016), random spatial placement of the nodes can occur. Therefore, random event boundaries should model both static and random positional impacts on **Front-end System Set** discovery. A random distribution can argue to represent increases in model fidelity, namely how the system decisions affect real-world system operating conditions from a statistical viewpoint. The *Failure Trace* DNS data set presents a model that indicates a concentration of more address resolution connections to a fewer number of nodes, Pang et al. (2004). As seen in the supplemental section on reliability, Section A.1.2, and random behaviour, section A.1.3, a simple random construct proposes the availability of the system to enrol a device at a point in time. Due to possible failed enrolment, a set of decisions on handling inbound/outbound **Front-end System Set** communication emerges. A key consideration for system control in this thesis is, of course; variability, non-queued transaction handling, finite interactions, and random load behaviours as indicated in Janert (2013).

By inverting the use case focus extracted from the concrete control problem in Janert (2013), the characteristics of a random set of automated **Front-end System Set** enrolments still exist. This inversion places the **Front-end System Set** at the core of the scaling argument as opposed to placing the number of servers that need to scale in order to meet inbound requests. Essentially, the placement of random **Front-end System Set** nodes for **Data Transmitting** compares to the just-in-time view of server scaling required for inbound requests.

Random failure behaviour exhibited within the **Front-end System Set** will also behave dynamically from the point that the first connection after enrolment. Based on the principle of a failed state within the *Survivorship* function, at some time in the measurement period, a random failure will occur, as shown in the supplemental Figure A.2. In setting the discovery principles, a definition of interactivity with **Front-end System Set** devices sees a system classification aimed specifically at identification, the variability of scale, and adaptivity impact

on reliability. As seen in Khaitan and McCalley (2015), the characteristics define parameters already discussed in preceding chapters. In the design techniques survey, Khaitan and McCalley (2015) outlines a comprehensive classification of system characteristics that could impact [Front-end System Set](#) discovery, such as;

1. the enablement of digitally accessible characteristics of physical objects,
2. high levels of automated interactions,
3. large and variable scales of connectivity,
4. large and variable temporal and spatial scales, and
5. self-organization and reconfiguration. The reconfiguration concept referenced by Khaitan and McCalley (2015) outline a basic dependability characteristic that incorporates a definition of reliability as seen in Marwedel (2010) and Aubrun et al. (2010).

When analysing these characteristics against the CPS gap mentioned in Mordecai and Dori (2017a), the following table applies when dealing with device discovery in the IoT-Ce. Adapted from Table 5.2, Table 5.3 shows four aggregate error handling requirements to

Table 5.2: Logical-Function Gap Characteristics adapted from Khaitan and McCalley (2015) and Mordecai and Dori (2017a)

Characteristic	Gap
Able to access physical object characteristic	Misaligned state
Automated interactions	Inability to interface
Large scales of connectivity	Inability to interface
Variable scales of connectivity	Inability to access defined configuration
Large temporal and spatial scales	Inability to interface
Variable temporal and spatial scales	Inability to access defined configuration

support a failed system state to be handled by multiple subprocesses. These errors identify with the gap handling classification outlined in the class and CPG types in Mordecai and Dori (2017a), and adapted in Table 5.2. Table 5.3 sequences in a possible order of existence where interfacing is primary.

The logical order for handling the error during enrolment is; (1) attempting to establish a

Table 5.3: ED Enrolment Error Set

Error Handling Code	Classification
ISARC-EH1	Inability to interface
ISARC-EH2	Misaligned state
ISARC-EH3	Inability to access defined IoT-Ce configuration
ISARC-EH4	Inability to obtain IoT-Ce runtime configuration

connection ([Connection Establishing](#)) and being unable to interface the [Front-end System](#)

Set, (2) establishing a connection and receiving a misaligned state message, (3) attempting to establish connectivity and being unable to access a defined configuration and (4) establishing connectivity and being unable to retrieve the runtime configuration. A simplified state-of-the-art OPD appears in Figure 5.4, coupled with the associated OPL and DSM. The OPD expands by including state handling for the inability to interface with and misaligned state errors for the Front-end System Set.

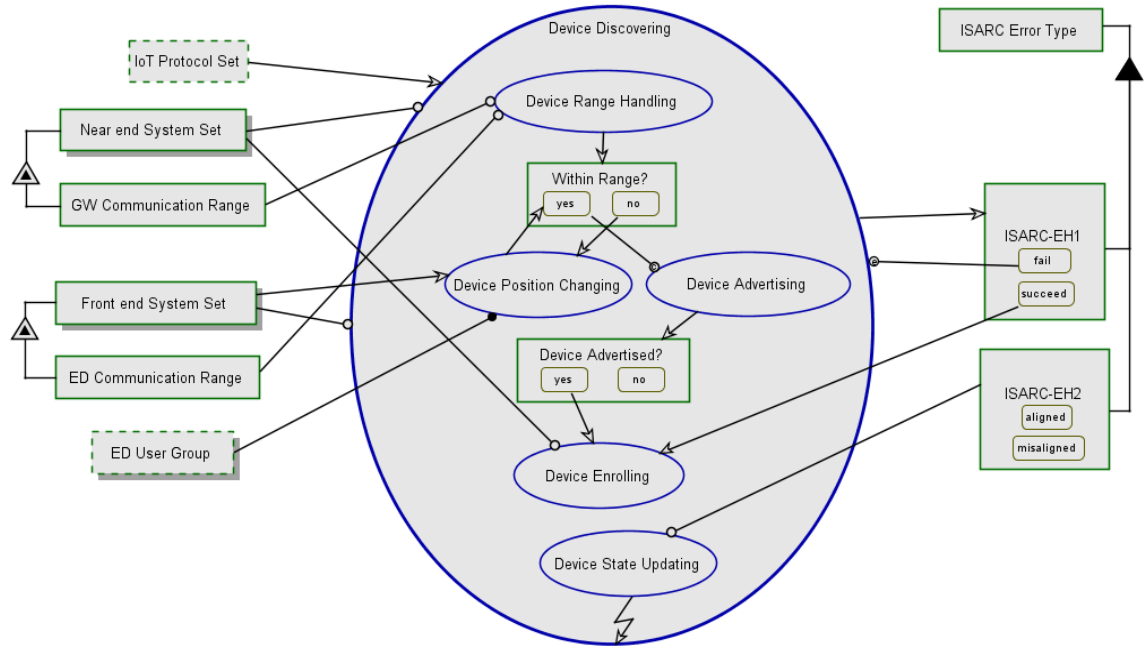


Figure 5.4: Function:OPD Device Discovering Function

Listing 5.2: Function:OPL Device Discovering Function

```

1 IoT Protocol Set is environmental.
2 Front end System Set is physical.
3 Front end System Set exhibits ED Communication Range.
4 Near end System Set is physical.
5 Near end System Set exhibits GW Communication Range.
6 ED User Group is environmental and physical.
7 ED User Group handles Device Position Changing.
8 ISARC Error Type consists of ISARC-EH1 and ISARC-EH2.
9   ISARC-EH1 can be fail or succeed.
10   ISARC-EH1 triggers Device Discovering when it enters fail.
11   ISARC-EH2 can be aligned or misaligned.
12 Device Discovering exhibits Within Range? and Device Advertised?.
13 Device Discovering consists of Device Position Changing, Device Range
14   Handling, Device Advertising, Device Enrolling, and Device State Updating.
15 Device Discovering requires fail ISARC-EH1, Front end System Set, and Near end System Set.
16 Device Discovering consumes IoT Protocol Set.
17 Device Discovering yields ISARC-EH1.
18 Device Discovering zooms into Device Range Handling, Device Advertising,
19   Device Position Changing, Device Enrolling, and Device State Updating, as well as Device Advertised? and
20   Within Range?.
21   Device Advertised? can be yes or no.
22   Within Range? can be yes or no.

```

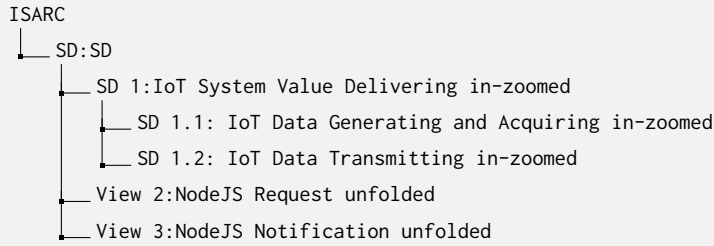
22	Device Range Handling requires ED Communication Range and GW
23	Communication Range.
24	Device Range Handling yields Within Range?.
25	Device Advertising occurs if Within Range? is yes.
26	Device Advertising yields Device Advertised?.
27	Device Position Changing changes Within Range? from no to yes.
28	Device Position Changing consumes Front end System Set.
29	Device Enrolling requires Near end System Set.
30	Device Enrolling consumes yes Device Advertised? and succeed ISARC-EH1.
31	Device State Updating requires ISARC-EH2.
32	Device State Updating invokes Device Discovering.

Table 5.4: DSM:Device Discovering Decision

Activity	A	B	C	D	E	F	G
Obj:NE System Set	A	o					
Obj:FE System Set	B	x	x	x	x	x	x
Proc:Range Handling	C	x					
Proc:Position Changing	D	x					
Proc:Device Advertising	E	x					
Proc:Device Enrolling	F	o	x				
Proc:Device State Changing	G	o	x				

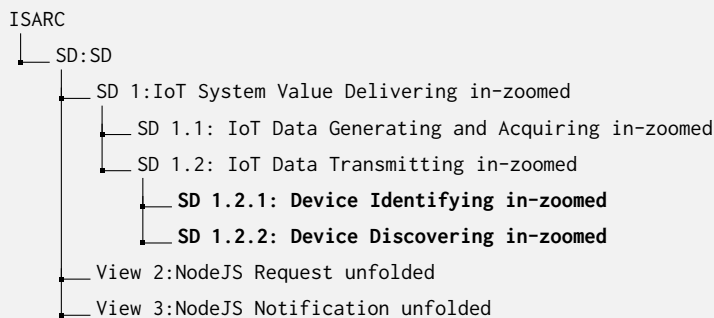
The ISARC system model challenges the OPD hierarchy in OPCAT to identify a suitable process location for the SDx that represents [Device Discovering](#) and associated [Device Auto Enrolling](#) processes. The critical systems definition question is; what is the primary value operand for enrolling a device into the IoT-Ce? If [Device Discovering](#) and [Device Auto Enrolling](#) locates as an in-zoomed process in SD1.1 [IoT Data Generating and Acquiring](#), it effectively means that enrolment is one of the main sub-processes that support [IoT Data Generating and Acquiring](#). It could also be the final process in SD 1.1. If it does occur as the final process in SD 1.1, the usability of enrolment could tie to most [IoT Data Generating and Acquiring](#) processes. This claim is not entirely true, as current [Front-end System Set](#) device functionality is built such that a device user can sense and collect sensor readings without being enrolled to an IoT gateway or a larger communications network. The user can execute [IoT Data Generating and Acquiring](#) processes directly using the [Front-end System Set](#) and thus the process is mostly decoupled. Unless a client application specifically binds its runtime sensor behaviour, the sensor largely decouples from being connected within the IoT-Ce. A simple test is to launch a sensor-based service application with mobile and WiFi services turned off on a mobile device. Readings for accelerometers, gyroscopes, and ambient light do not require a [Connected Front-end System Set](#). It does, however, require a [Powered Front-end System Set](#).

The ISARC initial hierarchy appears as;



A possible candidate for the placement of the SDx for [Device Discovering](#) and associated [Device Auto Enrolling](#) processes could be SD 1.2 [IoT Data Transmitting](#). It is clear that connectivity is a pre-existence requirement for transmitting data. In [Communication](#), there exists a requirement to understand both origin and destination, even in cases where asynchronous [Communication](#) is the primary message flow. Without prematurely creating a new domain for device discovery, the existing IoT domains for *Generation*, *Communication*, and *Analytics*, can continue to remain in their hierarchy location. One can, however, question the primary beneficiary value of [Device Discovering](#). One possible problem occurrence for this consideration is; *IoT-Ce System efficiency declining*, for which [Device Discovering](#) can change the state of [heterogeneous device enrolment efficiency](#) from [low](#) to [high](#). The auto-generated OPL for the [Device Discovering](#) function is listed below. Some of the critical discovery elements for error handling appears in the OPD. These are primarily objects representing an aggregation of error types while showing interaction with the primary discovery process. Under other modelling methods, these occur as two separate diagrams, namely class diagrams, and sequences. Secondly, an ordered sub-process flow from device range handling to a final device state update occurs in the [Device Discovering](#) SD.

The updated ISARC hierarchy shows both [Identifying](#) and [Discovering](#) as sub processes of the main [SD 1.2 IoT Data Transmitting](#) process.



5.2.3 Device Managing

Section 5.2.2 provides a brief explanation of the enrolment of the **Front-end System Set** as per guidance from the IoT Architecture Reference Model. The generalised set of definition-to-OPM extracted parameters produce the outline for a suitable parameter set that models a suitable attachment scenario for the auto-enrolment type. The management of the **Front-end System Set** now establishes a set of processes that could influence the primary enrolment flows. This section deals with device management in so far as access to the **Front-end System Set**. Device management focuses on the management of aggregate resources in **Front-end System Sets**. The exposure of abstracted system APIs produces the ability to model converged ED and GW capability, sections 2.2.2.1 and 3.4.2. The aforementioned presupposes that high-level programming languages run both server and client-side components. This presupposition is true for the application of IoT device integration.

The model extension shown in section 5.2 provides the initial single boundary architecture in Figure 5.2. However, in device management, configuration variations generate out of the simple structure to outline onboard versus tethered sensors. Onboard handling of sensors exhibits different boundary conditions when compared to tethered sensors. Tethered sensors have the possibility of a lossy communication channel between the **Front-end System Set** and sensor node. Recall the case of the Bluetooth application used for thermal scanning of possible breast cancer threats mentioned in a preceding chapter.

A sample device configuration variation appears in Figure 5.5. In the sample device, both sensors will fail to initiate a state call back from the NodeJS server on the **Front-end System Set** due to a user-initiated interrupt. Thus, the system boundary for a reachable state measurement is marred by another **Front-end System Set**'s software component inability to accept or produce requests. In the example below, the **Processing** capability indicated in Bera et al. (2016), can reform in an ISARC SD process as **Process Executing**. The process

Table 5.5: Sample Device Parameters

Path	Candidate Param.	Final State
FE-SS⇒Process Interrupt	SIGINT	Server Not Accessible
FE-SS⇒Process Interrupt	SIGBREAK	Server Not Accessible

interrupt required to create a faulted event in the system uses the *SIGINT* constants from the Node JS OS library. Figure 5.6 represents the systems architecture SD1 for this low-level process.

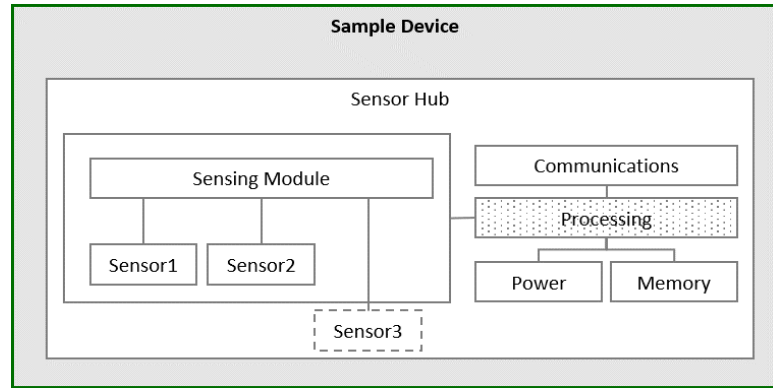


Figure 5.5: Sample Device Concept Summary. Adapted from Bera et al. (2016)

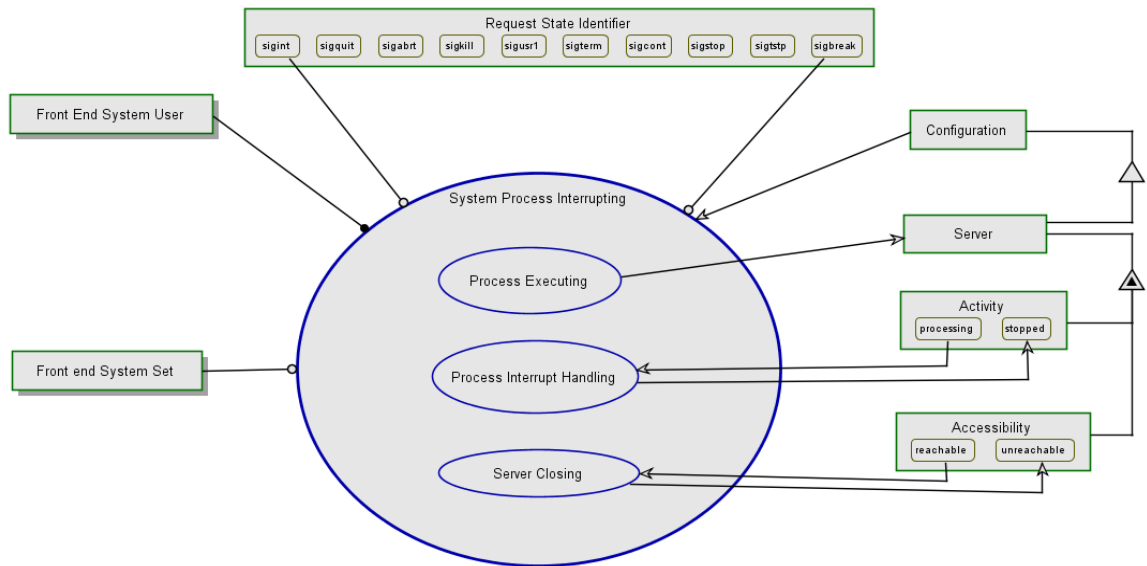


Figure 5.6: Function:OPD System Process Interrupting

Listing 5.3: Function:OPL System Process Interrupting

- 1 Front end System Set **is** physical.
- 2 Request State Identifier can be sigint, sigquit, sigabrt, sigkill, sigusr1, sigterm, sigcont, sigstop, sigtstp, **or** sigbreak.
- 3 sigkill lasts 5 milliseconds to 5 milliseconds.
- 4 Front End System User **is** physical.
- 5 Front End System User handles System Process Interrupting.
- 6 Server **is** a Configuration.
- 7 Server exhibits Activity **and** Accessibility.
- 8 Activity can be processing **or** stopped.
- 9 Accessibility can be reachable **or** unreachable.
- 10 System Process Interrupting consists of Process Executing,
- 11 Process Interrupt Handling, **and** Server Closing.
- 12 System Process Interrupting requires Front end System Set, sigint Request State Identifier, **and** sigbreak Request State Identifier.

```

13 System Process Interrupting consumes Configuration.
14 System Process Interrupting zooms into Process Executing,
15 Process Interrupt Handling, and Server Closing.
16 Process Executing yields Server.
17 Process Interrupt Handling changes Activity from processing to stopped.
18 Server Closing changes Accessibility from reachable to unreachable.

```

A NodeJS code snippet that represents the interrupt process in Figure 5.6 extracts from Cantelon et al. (2014)(pp.75 and 326). This code block labelled NodeJS OS Constant Emitter shows the initiation of an HTTP server using NodeJS. This server initiates on port 7001. The code block further indicates that the process executes with a specific identifier assigned by the operating system. The process identifier serves the purpose of sending kill commands for another state emitter like *SIGBREAK* or *SIGKILL*. A visual representation of the constant emitter code block execution order appears in Figure 5.7. Line 10 contains the execution of the Accessor Constant Emitter once the user executes a Ctrl-C keystroke.

Listing 5.4: NodeJS OS Constant Emitter

```

1 var http = require('http');
2 var server = http.createServer(function(req, res){
3   res.end('ISARC Accessor Constant Emitter');
4 });
5 server.listen(7001, function (err) {
6   console.log('Accessor is listening on port 7001');
7   console.log('The process ID is ' + process.pid);
8 });
9
10 process.on("SIGINT", function () {
11   console.log('Process Interrupt (SIGINT) Detected');
12   server.close();
13 });

```

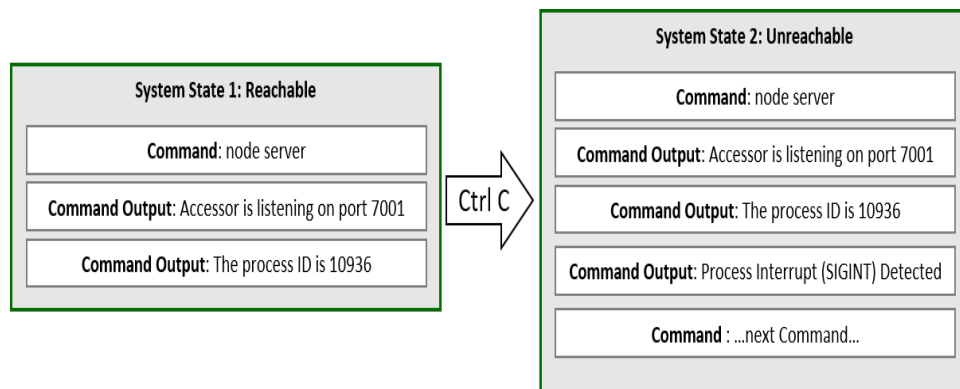


Figure 5.7: NodeJS SIGINT Emitter Output

In summary, Figure 5.5 outlines a fairly basic view of a sensor node architecture. Once overlaid with the complexity of an actual implementation using the OS constant variations found in Android, active-learning questions come to light such as; Does the [Processing](#) execute in a single boundary, or does concurrency requirements in physical sensing mandate an alternate

5.3 Model Validation Evidence

The model evidence selects one of the more complex SDs to present the optionality of model parameter selection. The supporting object and process, **IoT System Controller** candidate, seen in Chapter 4 Figure 4.6, provides a concrete implementation of the control control problem in the **Front-end System Set** client-server communication case.

The testing capabilities of OPM indicates that the first simulated step for **IoT System Control Handling** does not initiate the **Queued Request Purging** immediately. Hence, a **queued request** does not exist at the time of the first **control action**. Table 5.6 reproduces the OPCAT 4.2 life-span graph for the first 3 testing steps of the system diagram in Figure 5.8. The solid red circles in Figure 5.8 traverse the interaction links between processes and objects and change the state of objects along the execution path. The model execution in a model as large as Figure 5.8 requires meticulous planning. Incorrect usage of links and objects will result in OPCAT simulation failure caused by missing agent information or malformed links. The table below is reproducible in OPCAT using the OPD in Figure 5.8. The solid red circles in Figure 5.8 represent a point in time view; however, these circles

Table 5.6: IoT System Controller Test Step 1 to 3

Name	Type	Step:1	Step:2	Step:3
...	...	...	...	...
Queued Request Purging	Process	not active[1]	active[1]	not active[1]
Queued Request	Object	accessible[1]	exists[1]	purged[1]
...	...	...	...	...

move dynamically during simulation in OPCAT 4.2. The focus of the first model test is handling a queued purge request - **Queued Request Purging**. The model can alternatively test a viewpoint such that **System Request Handling** process is the focus and not the controller itself. In this case, a request that enables the parameter range **state:out of range**, immediately activates the controller. The **IoT System Control Handling** process activates immediately in step 2 after the **System Request Handling** process completes. Figure 5.9 indicates an updated activation flow for the same model, with a different initiation point, namely **System Request Handling** first, which instructs **IoT System Control Handling** process. The associated 3 step test shows the parameter range changing from **out of range** to **in range**. Table 5.7 shows a recreated OPCAT life-span graph indicating a parameter range as **out of range** in step 1, the **System Control Handling** process as **not active** in step 1, and then **active** in step 2 *without manual adjustment of an object state* from active to inactive in the SD. This can be seen through the executable model test (red dots) and shaded objects in Figure 5.9.

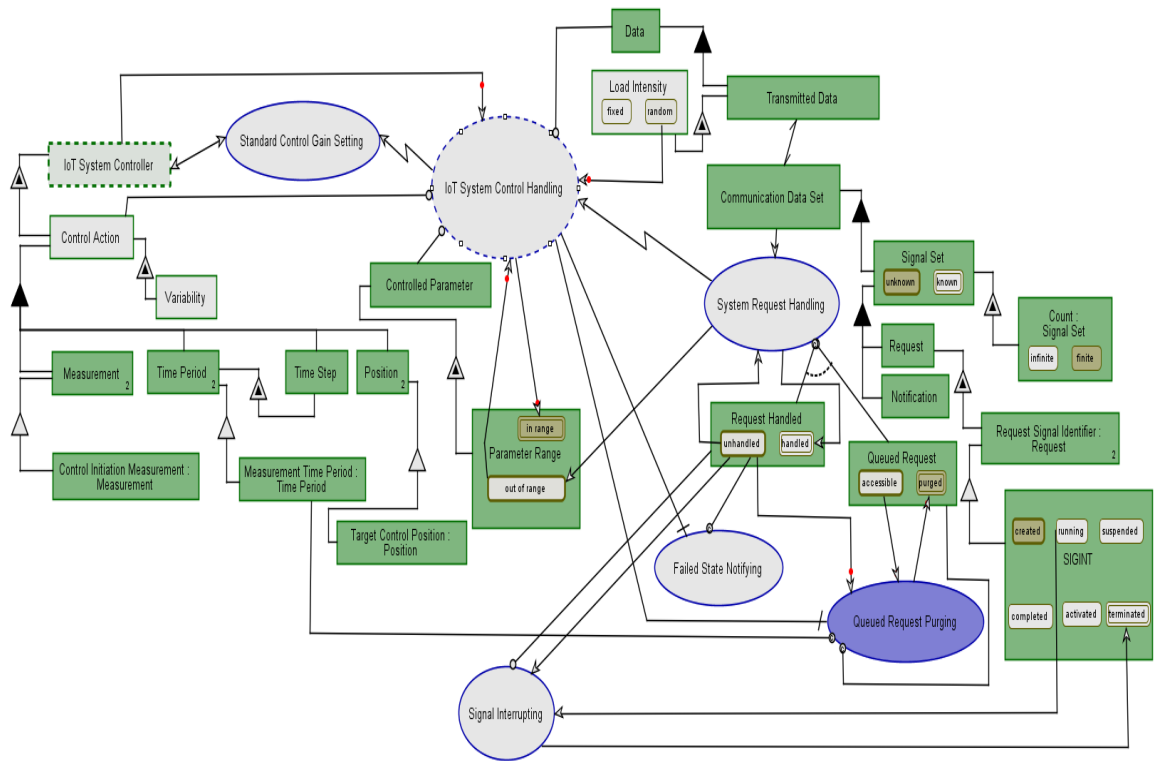


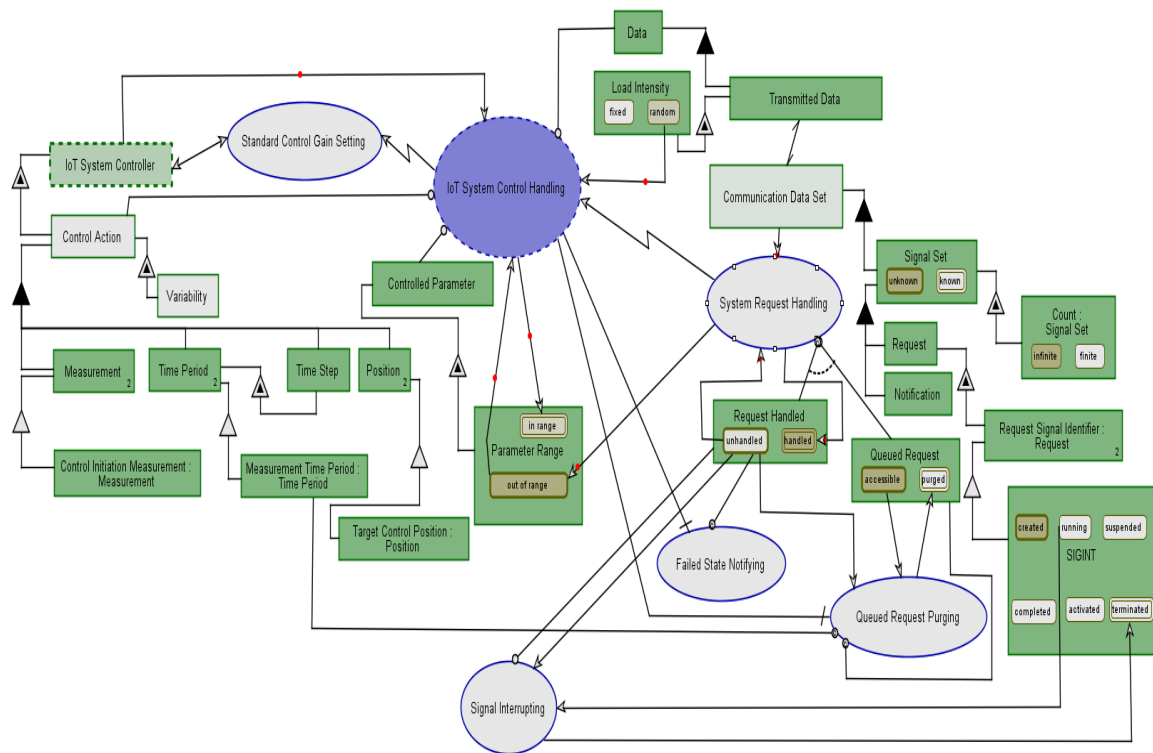
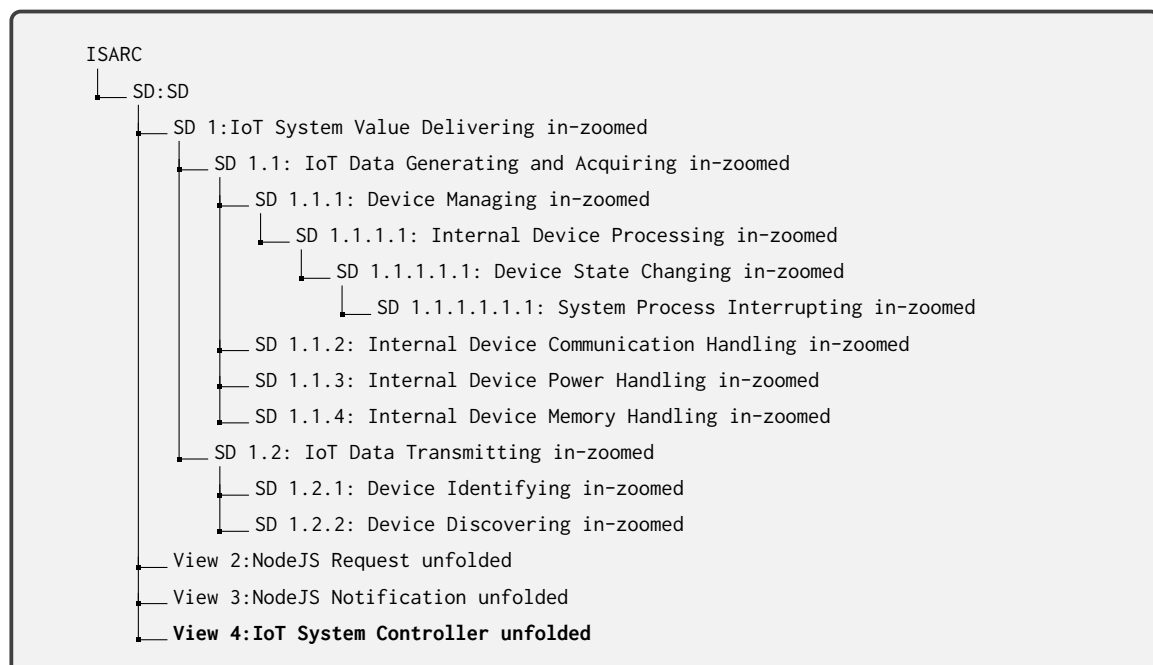
Figure 5.8: IoT System Controller Running Test Step 1 to 3

Table 5.7: IoT System Controller Test Step 1 to 3 Alternate Initiation Point

Name	Type	Step:1	Step:2	Step:3
...	...	...	...	...
Parameter Range	Object	out of range[1]	out of range[1]	in range[1]
IoT System Control Handling	Process	not active[2]	active[2]	not active[2]
...	...	...	...	...

The resultant objects and states from this section present as an unfolded, non-sequential process that can apply at any level in the hierarchy of the system. The primary source invocation in Figure 5.9 is the [System Request Handling](#) process, however the invoking process could also be outbound from the [Device Identifying](#) process towards the [IoT Control System Handling](#) process. The system model's invocation link can be effortlessly changed to reflect a link between Figure 5.9 and the earlier Figure 4.5.

Finally, the representation of the modelled control problem links through the primary unfolded structure OPD into a candidate systems hierarchy. An extension of the earlier ISARC hierarchy shows the integration of the unfolded [IoT System Controller](#) from earlier Figures 4.4, 4.5, and 4.6 as an unfolded testable structure that can attach at any layer within the hierarchy.



5.4 SDn Model Extension Summary

The SDn model extensions present an n-level decomposed diagram set that enables a simulation-ready validation of the ISARC. The original primary SD, SDx, and SDx.x diagrams remain intact while additional sub-level diagrams append to the required nodes in the SD hierarchy. These nodes appear in SDx.x.x, SDx.x.x.x, SDx.x.x.x.x and then finally an application-level state change in-zoomed at SDx.x.x.x.x.x. Additionally, an unfolded multi-use controller set attaches to the main SD, which is usable at any hierarchy level, as mentioned above. Model compliance of the ISARC to OPM ISO-19450 emerges through the use of standard OPM symbols that can validate using runtime model outputs.

Figure 5.10 presents an overview of the model extensions and the associated impact of model feedback of lower-level SDn representations on the primary system purpose. Thus, the primary SD decomposes into SDx, SDx.x and towards the final device operating system function in SDx.x.x.x.x.x. The state change for the overall runtime configuration of the IoT-Ce **Front-end System Set** to an interrupted state negatively impacts the initially planned system value of changing; **dependability** from **low** to **high**. To recall, the primary SD node represents the planned system value of changing **dependability** from **low** to **high**.

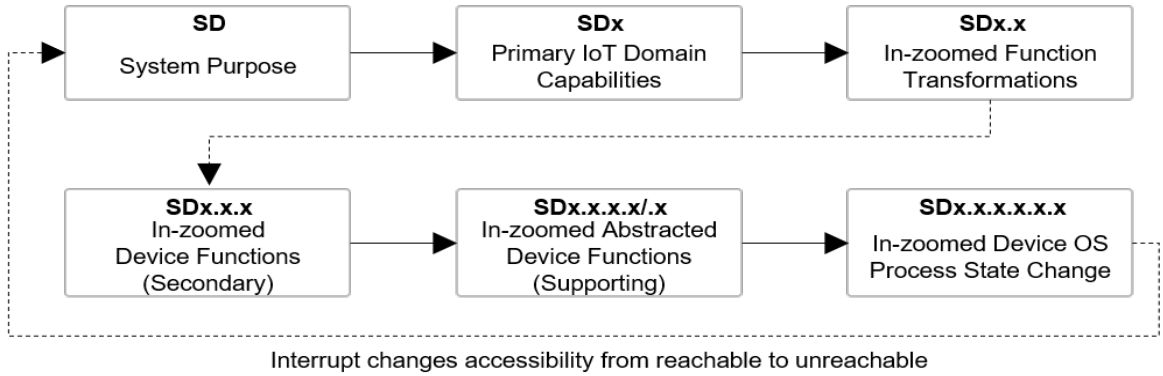


Figure 5.10: System Model Extension Summary

Although the specific interrupt process in SDx.x.x.x.x.x can present arguments from the point of completeness, the value of modelling the ISARC using OPM naturally supports an extensible model that can guard against the impacts of model rework, as exhibited in the ISARC hierarchy evolution. An example of the extensibility of the model is present in the **IoT System Control Handling** process in the model validation evidence section. The **IoT System Controller** SD shows the extent to which the object can incorporate into a larger model requirement alongside **IoT System Control Handling**. For brevity, the application of a concrete control problem applies to the **IoT System Controller** model. Furthermore, on

the rework of the model's objects and processes, it is found that inaccurate assumptions on the inputs of the process generally cause rework, Eppinger and Browning (2012)(pp.135). However, refinement of diagrams for clarity and completeness, as per Dori (2011), appears on several occasions in the preceding chapters. Thus, the interrupt process reflects a plausible change of state, namely an interrupt of the IoT-Ce **Front-end System Set** application, which results in an unreachable device resource at a point in time.

The consideration for change to the architecture is understanding the magnitude of interdependent system change while ensuring minimisation of architectural shifts. This understanding should allow the system to settle in a configuration that is suitable for the system. A stable set of configurations agrees to the objective functions of reducing feedback guided by reduced assumptions in Process DSMs, Eppinger and Browning (2012)(pp.141). The split between a need for closed-loop feedback in a system and the decisions that impact feedback in a system are linked in the ISARC decision model.

5.5 SDn Decision Model Validation

The section on the Decision Model Validation serves as the **secondary contribution** to the hypothesis that *low-level IoT-Ce configurations that influence decisions about dependability can abstract into a conceptual IoT system architecture to support model validation activities*. Claims in the preceding chapters show that defining an IoT-Ce parameter set enables a model that structures into a natural hierarchy when discussing object (system form) and process (system object transformation). The definition-to-OPM approach suggests using OPM ISO 19450. In parallel, Dori's core systems realisation questions; *Problem, Purpose, Primary Function, Enabler*, and *Environment* frames the system boundary that yields exciting results for ongoing system change and arguments. The ISARC decision model based on DSM and applied in Python exists as a final validated viewpoint of ISARC variations.

5.5.1 Model Validation Implementation

The validation of the ISARC uses Python to provide a reusable component structure for supporting ongoing ISARC architectural decisions. The ISARC model validation uses Python due to Python's proximity to simulation and data science functions in academia and industry. As seen in the Python libraries by DeNero et al. (2020), the efficient handling of simulation, sampling, distribution, array, and visualisation functions is possible. These functions are notably present in the Python component set built to support the online data science course material by the University of Berkeley, Adhikari et al. (2020). The import of a historical data set serves as the basis of the validation test case. The DNS Failure Trace

Archive data set by Pang et al. (2004) and published by Epema et al. (2014) - handles in several different technologies. Technologies such as direct database import using the FTA SQL dataset, comma-separated value handling in Excel, or a command-line shell script are all possible. Additionally, Epema et al. (2014) also produces Matlab and a simple sample read and print script in Python.

The ISARC suggests using CSV imports in Python due to Python's inherent flexibility and ongoing data science activities. Finally, for browsable results visualisation, Plotly Technologies Inc. (2020) produces a standard visualisation library that presents a browsable heatmap with the same form as an n-squared matrix. Other inline graphs use standard *matplotlib* graph projections. A reproduction of the system configuration decisions models found in DSMs 1 to 4 in Chapter 4, section 4.6, is shown as;

1. **Model 1** - Known IoT Subsystem Interactions as Parameter Decision Model 1,
2. **Model 2** - Perceived IoT-Ce OPM as Parameter Decision Model 2,
3. **Model 3** - Known Device OS Signalling Constants as Parameter Decision Model 3,
and
4. **Model 4** - Known Process Interactions as Parameter Decision Model 4.

5.5.2 ISARC DSM Decision Model 1 to 4 Initialisation

The initialisation of the DSM for ISARC Model 1 indicates the subsystem interactions within a **Front-end System Set**, and between edge devices and IoT Gateway. A total of nine components are shown across two subsystems within a single system boundary, as indicated in Chapter 4, Table 4.9. The code block below presents the *NumPy* array for the ISARC Model 1. *NumPy* is a standard Python library, Oliphant (2006). Note the array diagonal as having a single constant value of 1. The variable parameter indicator for ISARC Model 1 (M1.x) in the initialised code block below consumes the definition-to-OPM references from preceding sections and chapters to ensure parameter consistency. The variable parameter indicator value aims to represent a unique instance of the parameter.

Listing 5.5: Code Block 1: ISARC Decision Model 1 Initialisation

```

1 #Decision Model 1: IoT-Ce Subsystem Interactions as indicated in Chapter 4 Table 4.9
2 #Filename: isarcDecisionModel1Init.ipynb
3 import numpy as np
4
5 dm1Param = [
6     ('M1.C1.1','Communication'),('M1.C1.2','HardwareAbstraction'),
7     ('M1.C1.3','OperatingSystem'),('M1.C1.4','RemoteManagement'),
8     ('M1.C2.1','DataManagementMessaging'), ('M1.C2.2','Communication'),
9     ('M1.C2.3','ApplicationRuntime'), ('M1.C2.4','OperatingSystem'),

```

```

10  ('M1.C2.5','RemoteManagement')]
11  dm1 = np.identity(9)
12  decisionModel1 = np.concatenate((np.asarray(dm1Param),dm1), axis=1)
13  decisionModel1
14
15  #Save for Global Variable use
16  np.save('DecisionModel1Property.npy',decisionModel1)
17
18  #Redacted output example for component parameter array concatenated with
19  #NumPy Identity function
20  #array([
21  #['M1.C1.1', 'Communication'          , '1.0', '0.0', '0.0', '0.0', ...],
22  #['M1.C1.2', 'HardwareAbstraction'    , '0.0', '1.0', '0.0', '0.0', ...],
23  #['M1.C1.3', 'OperatingSystem'        , '0.0', '0.0', '1.0', '0.0', ...],
24  #['M1.C1.4', 'RemoteManagement'      , '0.0', '0.0', '0.0', '1.0', ...],
25  #['M1.C2.1', 'DataManagementMessaging', '0.0', '0.0', '0.0', '0.0', ...],
26  #['M1.C2.2', 'Communication'          , '0.0', '0.0', '0.0', '0.0', ...],
27  #['M1.C2.3', 'ApplicationRuntime'     , '0.0', '0.0', '0.0', '0.0', ...],
28  #['M1.C2.4', 'OperatingSystem'        , '0.0', '0.0', '0.0', '0.0', ...],
29  #['M1.C2.5', 'RemoteManagement'      , '0.0', '0.0', '0.0', '0.0', ...]]
30  #, dtype='<U32')

```

The output of the *isarcDecisionModel1Init.ipynb* codeblock produces a DSM-typed matrix with constant diagonal values. The row and column intersection holds a constant value that can be locked using Python. This will ensure that the value becomes immutable. Oliphant (2006)(pp.48) suggests the use of *flags.writeable = False*.

Listing 5.6: Code Block 2: ISARC Decision Model 2 Initialisation

```

1  #Decision Model 2 IoT-Ce OPM as indicated in Chapter 4 Table 4.10
2  #Filename: isarcDecisionModel2Init.ipynb
3  import numpy as np
4
5  dm2Param = [
6  ('M2.1','SD, SD1','Process','IoT Edge System Value Delivering'),
7  ('M2.2','SD, SD1','Object','IoT System User Group'),
8  ('M2.3','SD','Object','IoT System Set'),
9  ('M2.4','SD, SD1','Object','Near End System Set'),
10 ('M2.5','SD','Object','NE System Dependability'),
11 ('M2.6','SD, SD1, SD1.1','Object','Front End System Set'),
12 ('M2.7','SD, SD1','Object','FE System Dependability'),
13 ('M2.8','SD, SD1','Object','IoT System Controller'),
14 ('M2.9','SD1','Process','IoT Data Generating and Acquiring'),
15 ('M2.10','SD1','Object','Sensed Data'),
16 ('M2.11','SD1','Process','IoT Data Transmitting'),
17 ('M2.12','SD1','Object','Transmission Attempt'),
18 ('M2.13','SD1','Process','IoT Data Analysing'),
19 ('M2.14','SD1','Process','IoT System Control Handling'),
20 ('M2.15','SD1','Object','Edge Device'),
21 ('M2.16','SD1','Object','Sensing Capability'),
22 ('M2.17','SD1','Object','Communication Capability'),
23 ('M2.18','SD1','Object','Configuration Set'),
24 ('M2.19','SD1','Object','System State Data Set'),
25 ('M2.20','SD1.1','Process','Device Identifying'),
26 ('M2.21','SD1.1','Object','Identification'),
27 ('M2.22','SD1.1','Object','Assigned Address'),
28 ('M2.23','SD1.1','Process','Device Connection Establishing'),
29 ('M2.24','SD1.1','Object','Device Connection State'),
30 ('M2.25','SD1.1','Process','Device Data Sending'),
31 ('M2.26','SD1.1','Process','Connecting Terminating')]
32 decisionModel2 = np.concatenate((np.asarray(dm2Param),np.identity(26)),
33 axis=1)

```

```

34 print(decisionModel2)
35
36 #Save for Global Variable use
37 np.save('DecisionModel2Property.npy',decisionModel2)

```

Similarly, the code block for the initialisation of ISARC Decision Model 2 to 4 indicates the replication of the *NumPy* concatenation operation using the Python *Identity* function. Oliphant (2006)(pp.89) indicates the *Identity* function to “return a 2-d array of shape (n,n) and data type, dtype with ones along the main diagonal”. To provide a suitably paced walk-through of the Python approach for the ISARC, the usage of variables such as the length of the parameter array is directly declared using a defined numeric value in earlier initialisation code blocks. However, further initialised decision models use a dynamic approach. As seen in Code Block 3: ISARC Decision Model 3 Initialisation, the calculation of the array length applies *np.identity(len(«Parameter Array»))*. The output of this function will automatically scale the number of array items and provide the requisite dynamism to the ISARC decision models.

Listing 5.7: Code Block 3: ISARC Decision Model 3 Initialisation

```

1 #Decision Model 3 Device OS Signalling Constants as indicated in Chapter 4 Table 4.11
2 #Filename: isarcDecisionModel3Init.ipynb
3 import numpy as np
4
5 dm3Param = [
6     ('M3.1','Notify','SIGHUP'),
7     ('M3.2','Request','SIGINT'),
8     ('M3.3','Request','SIGQUIT'),
9     ('M3.4','Notify','SIGILL'),
10    ('M3.5','Notify','SIGTRAP'),
11    ('M3.6','Request','SIGABRT'),
12    ('M3.7','Request','SIGIOT'),
13    ('M3.8','Notify','SIGBUS'),
14    ('M3.9','Notify','SIGFPE'),
15    ('M3.10','Request','SIGKILL'),
16    ('M3.11','Request','SIGUSR1/2'),
17    ('M3.12','Notify','SIGSEGV'),
18    ('M3.13','Notify','SIGPIPE'),
19    ('M3.14','Notify','SIGALRM'),
20    ('M3.15','Request','SIGTERM'),
21    ('M3.16','Notify','SIGCHLD'),
22    ('M3.17','Notify','SIGSTKFLT'),
23    ('M3.18','Request','SIGCONT'),
24    ('M3.19','Request','SIGSTOP'),
25    ('M3.20','Request','SIGTSTP'),
26    ('M3.21','Request','SIGBREAK'),
27    ('M3.22','Notify','SIGTTIN'),
28    ('M3.23','Notify','SIGTTOU'),
29    ('M3.24','Notify','SIGURG'),
30    ('M3.25','Notify','SIGXCPU'),
31    ('M3.26','Notify','SIGXFSZ'),
32    ('M3.27','Notify','SIGVTALRM'),
33    ('M3.28','Notify','SIGPROF'),
34    ('M3.29','Notify','SIGWINCH'),
35    ('M3.30','Notify','SIGIO'),
36    ('M3.31','Notify','SIGPOLL'),
37    ('M3.32','Notify','SIGLOST'),

```

```

38 ('M3.33','Notify','SIGPWR'),
39 ('M3.34','Notify','SIGINFO'),
40 ('M3.35','Notify','SIGSYS'),
41 ('M3.36','Notify','SIGUNUSED')]
42 decisionModel3 = np.concatenate((np.asarray(dm3Param),
43 np.identity(len(dm3Param))), axis=1)
44 print(decisionModel3)
45
46 #Save for Global Variable use
47 np.save('DecisionModel3Property.npy',decisionModel3)

```

Code block *isarcDecisionModel3Init.pynb* applies the Android OS Constant array structure as the baseline for ISARC Decision Model 3. The *NumPy* array enables a count of each classification found in Table 3.7. The code `len(decisionModel3[np.where(decisionModel3=="Request")])` yields a response of 11 and `len(decisionModel3[np.where(decisionModel3=="Notify")])` yields a response of 25. At the time of publication there are 36 unique signalling constants available, Linux (2018).

Greater usability of the data structure is available once the array translates to a *datascience* table. DeNero et al. (2020) applies the use of sort, group, and pivot functions.

Listing 5.8: Code Block 4: ISARC Decision Model 4 Initialisation

```

1 #Decision Model 4 Runtime Process Interactions as indicated in Chapter 4 Table 4.12
2 #Filename: isarcDecisionModel4Init.ipynb
3 import numpy as np
4
5 dm4Param = [
6 ('M4.1.A','Created','a'),
7 ('M4.2.B','Running','b'),
8 ('M4.3.C','Suspended','c'),
9 ('M4.4.D','Completed','d'),
10 ('M4.5.E','Activated','e'),
11 ('M4.6.F','Terminated','f')]
12 decisionModel4 = np.concatenate((np.asarray(dm4Param),
13 np.identity(len(dm4Param))), axis=1)
14 print(decisionModel4)
15
16 #Save for Global Variable use
17 np.save('DecisionModel4Property.npy',decisionModel4)

```

The establishment of ISARC Decision Model 1 through ISARC Decision Model 4 creates a programmable interaction set amenable to operating configurations, boundary conditions, and constraints appear. The bounded interaction set creates the primary system boundary such that architecture decisions segment using the structured hierarchy. For example, ISARC Decision Model 1 (DM1) contains components M1.C1.1 through M1.C2.5 with specific interactions. These would link to DM2, which represents a lower-level interaction set of the components in DM1. The DM2 further links to DM3 that exhibits operating system states of a **Front-end System Set**, which finally links to process states in DM4. DM4 represents operating system states through process transition. Thus, a hierarchy of system interactions applies from initial conceptual architecture to the intended process transitions. Finally, the

DSM structure enables native input column (IC) values where a decision weight will appear at the row and column intersection.

The next set of code components shows that hierarchy and system boundary play a role in assigning decision constraint scenarios for different configurations given the decision model and the earlier constraint set. Standard functions from the Python *NumPy* library appear in the code block below.

5.5.3 Model Validation Constraints

Indicated in Chapter 1, the ISARC contains several constraints concerning the **Front-end System Set**. These are;

1. The IoT-Ce system is regarded as *closed* once an IoT device becomes attached to a local or operator network. The ability of the EG/GW capable mobile smartphone, termed a Front-end System Set, to initiate communication under a constraint that all required components allow this communication. For example, firewall port configurations, device address advertising, and general availability. The notion of harder subsystem boundaries for edge components is noted as user-controlled front-end and near-end system sets and operator-controlled far-end system sets (Cloud), Yu et al. (2017).
2. **Front-end System Set** instances *increment independently in a non-sequential manner* from a single node towards the physical or logical upper limits of the DNS systems hierarchy. Local and remote DNS components implement a variable technical limit in terms of the number of hosts (Front-end system sets) it can respond to in parallel,
3. **Front-end System Set** instances are *enrolled automatically* using gateway initiated interactions. Partial and Manual enrolment is not explored in detail; however, for brevity, detail is included in Appendix B,
4. internal **Front-end System Set** processing delays have a *variable upper limit*.
5. **Front-end System Set** instances can enter into several *sleep and standby states* in between transmission, Rubenstein (2017). Thus, an optimisation strategy concerning scheduling of application processes is considered such as those indicated in Yu et al. (2017),
6. the request for **Front-end System Set** states from a requesting IoT application could *traverse a combination of network protocols* via distributed gateways and control subsystems, Yu et al. (2017). Yu et al. (2017) indicates that the shortest distance will

reduce transmission time. To reach a target **Front-end System Set**, increases in request-response round-trip time is *variable*, and

7. the processing of data for model validation purposes is limited to *simulated control*. It could include post-event analysis of events supporting the optimisation of control decisions. A standard Proportional and Integral (PI) control strategy applies to the simulated control proof.

The constraint model rules appear in individual proof code blocks later in this chapter. These constraints apply to the generation of ED/GW address connection attempts against the LDNS node population in the FTA data set.

5.6 Failure Trace Data Set

The positioning of runtime configuration exhibits its roots in the core understanding of system structure and behaviour. Understanding of configuration increases fidelity in systems models. Architecture fidelity represents a level of trust that a system will operate as expected while in operation. As indicated in the options for model verification and validation, section 2.7, parts and components undergo process simulation as a possible model type in early system design phases. Additionally, if these parts or components are whole products, they can undergo Failure Event Analysis (FEA) in this phase. Cameron et al. (2017) sees process simulation execution in the model stage as supporting ongoing systems engineering phases up to post-production. Furthermore, failure analysis and error correlation for distributed systems in research have shown interest in recent years. The research area produced a series of data sets located in an online *Failure Trace Archive* (FTA), Epema et al. (2014). These data sets support the analysis of error correlation across a distributed architecture for several classes of internet-connected system classes, Kondo et al. (2010), and specifically Domain Name Systems (DNS), Pang et al. (2004).

The representation of a DNS meets IoT-Ce protocol requirements and a system model for handling single system boundary issues, the composition of subsystems, and individual component reliability. Recent studies have shown that the system pattern for address mapping handles like a two-tiered problem with component 1 calling component 2 in subsystem 1, and then component 2 calling component 3 in subsystem 2, within the same systemic flow. Thus, the requirement for a process-based system identifies closely with failure analysis. Hence, a reference towards state-awareness in components is argued for - similar to those found in Yigitbasi et al. (2010). In Yigitbasi et al. (2010), the need for a class of state-awareness

entities in time-varying behavioural analysis of failures in large-scale distributed systems has vital benefits. State-awareness enables;

1. having “significant” impact on solutions that handle fault behaviour pro-actively,
2. having an impact on failure resolution strategies for systems that are in a failed state,
3. being aware of temporal conditions that exist for ‘smart checkpointing’ and system, subsystem, or component level scheduling, and
4. having information on creating adaptive fault-tolerant mechanisms.

These benefits are a call-back to the notion of state as a driver for adaptive systems design seen in IBM (2005), Cheng (2008), Swanson et al. (2014), Ashraf et al. (2016), and Villegas et al. (2017). The design consideration, which outlines the need for information for creating fault-tolerant mechanisms as seen in Yigitbasi et al. (2010) provides the required visibility, and by implication, awareness of the state of a system at any given point in time. A practical example of a set of failed state occurrences emerges from the FTA. The sample data set submitted by Pang et al. (2004) contains a total of 62,201 hosts. The data set indicates a higher number of failure occurrences in fewer DNS servers, as shown in Figure A.3.

The following section explains the structure of the FTA data set and produces graphical representations such as charts, histograms, and other significant statistics. Finally, inferences that present a series of patterns not stated in Pang et al. (2004) supports the ISARC model proof.

5.6.1 Failure Trace Archive Data Structure

The FTA data set is downloadable in comma-separated and database file formats. These formats enable efficient import and data handling possibilities. The data structure contains a series of specific table structures that are, for the most part, a normalised data structure. Epema et al. (2014) defines the entities in the data set as having the content that produces a single file output per entity. Standard keys across the files enable the relational handling of file content.

- A **Platform** entity contains descriptions of nodes. Examples of platforms are wide area networks, desktop computers, supercomputers, and web servers.
- A **Node** entity contains a set of components, which is either a software module or hardware resource of the node. Each node can have several components, such as CPU speed, memory parameters, or client availability, each with an associated trace event set.

- A **Component** entity describes attributes of the software module or hardware resource of a node.
- A **Component performance** entity describes the component performance, as measured through specific component benchmarks.
- A **Creator** entity contains the author responsible for the trace data set. This table stores details about citations and copyright of authors that have submitted FTA data sets and analysis.
- An **Event Trace** entity contains the trace of a series of events, with corresponding timing information.
- An **Event State** entity contains the state corresponding to an event trace. For example, for node availability, the event state could be the availability of the node. In Pang et al. (2004), the **Event State** entity is empty as the state of the node's availability is found in the actual **Event Trace** entity.

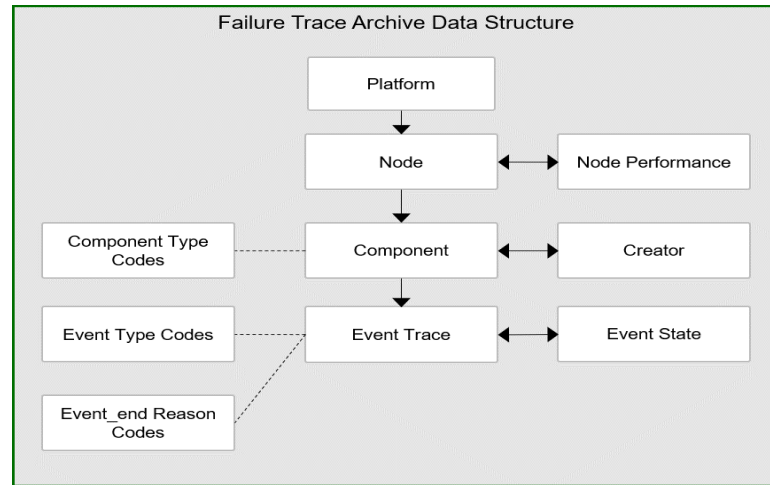


Figure 5.11: Failure Trace Archive Data Structure. Extracted from Epema et al. (2014)

The entity structure also contains component type codes, event type codes, and event end reason codes. These contain master reference data for serving primary key values into the component and event trace entities. The event type codes linked to the event trace entity identify the state of a node such as $\{0:\text{"Unavailability"}\}$ and $\{1:\text{"Availability"}\}$. The entity of specific interest in Figure 5.11 is the *event trace data set*. This data set contains the population of events and the associated state for each event for each failure trace source. Unlike the other entities in the data model, the event trace contains the actual measurements of a state instead of reference or descriptive metadata. Entities such as component

and node list the actual inventory of nodes in the event data sets. However, it is shown that all nodes identified by an IP address exist in both node and event trace data sets. Thus using a distinct summary of the nodes in the event trace data produces the total population of nodes present in the node entity.

The event traces for the Pang DNS data set contains specific measurements for **62201** nodes over a two-week measurement period in March 2004 to April 2004. There is a total population of **384991** measurement entries in the Pang DNS data set. The formal structure of the model elements is shown in Table 5.8. The primary fields in Table 5.8 uniquely

Table 5.8: DNS Availability Event Trace Structure extracted from Epema et al. (2014) and Pang et al. (2004)

Column	Description
event_id	A unique identifier of the event state
component_id	A unique identifier for specific component trace data
node_id	A unique identifier for event's node
platform_id	An identifier of the platform that represents the node's parent
node_name	The name of node
event_type	An event type. Events are characterised by 0 for unavailable nodes and 1 for available nodes
event_start_time	The start date and time of the event represented in UNIX epoch time
event_end_time	The end date and time of the event represented in UNIX epoch time
event_end_reason	The event type reason or state changed at the end of this trace. Event reasons such as infrastructure, hardware or software related codes are listed in the Event End Reason Code table.

identify specific event trace instances are node_name, event_type, event_start_time and event_end_time. The Platform, Node, and Component hierarchy, which identify with platform_id, node_id, and component_id in the FTA data structure, align with a multi-level system structure throughout the preceding chapters. For brevity, a System-Subsystem-Component hierarchy. For this reason, the identified node's IP Address is the identifier that uniquely identifies a single node in the system boundary. To recall, an Address identifies a node used in Addressing a Communication path for a node. The Address is a logical identifier that resides on the second level of the Platform-Node-Component hierarchy. The ISARC agrees with the structural aspects of the identified DNS ecosystem, which mimics the structure outlined for edge computing seen in preceding chapters.

5.6.2 FTA Data Set Handling

Python supports several libraries that enable efficient handling of arrays, data types, and field calculations. The import code block below indicates the first use of Python Pandas. Pandas is a data manipulation package. A Panda Dataframe is the data type for storing two-dimensional data. A good match for the n-squared matrix structure, but more importantly,

Pandas contain one of the most efficient import methods for handling delimited data sets. One could argue that more straightforward import methods exist; however, Pandas provides a head and tail method for the initial sampling of records in a large data set. In Python 3.7, the following code block imports the Pang et al. (2004) FTA DNS Failure Event data set.

Listing 5.9: Failure Trace Event Pandas Import

```
1 import pandas as pd
2 traceds = pd.read_csv("event_trace.csv")
3 traceds.head(7)
```

The resulting table for the first 7 records of this import is shown in Table 5.9.

Table 5.9: DNS FTA Data Set First 7 Records extracted from Pang et al. (2004)

event id	...	...	node name	event type	event start time	event stop time	event end reason
0	...	...	0.0.0.1	1	1080209426	1080308403	NaN
1	...	...	0.0.0.1	0	1080308403	1080315634	NaN
2	...	...	0.0.0.1	1	1080315634	1080318710	NaN
3	...	...	0.0.0.1	0	1080318710	1080336996	NaN
4	...	...	0.0.0.1	1	1080336996	1080343801	NaN
5	...	...	0.0.0.1	0	1080343801	1080376236	NaN
6	...	...	0.0.0.1	1	1080376236	1080404578	NaN

A thoroughly prepared code block that transforms the Epoch event start time and event stop time in the FTA DNS delimited file appears in a subsequent code block. As indicated, a minimal number of library imports are present to handle the FTA data set. The Python *NumPy* library is included in the import, however not required for the actual data set import. The *np* object supports NumPy operations and methods used later in the analysis. Similarly, the Python *datascience* library enables table structure handling that supports analysis operations for random and fixed sampling used later in the analysis of the FTA data set.

Therefore, the DNS Event Trace handling code block imports these libraries in preparation for further use of the operations from these libraries. A series of column transformations apply to the base FTA data set such that the representation of well-formed date fields and an event period appears in the new data set.

Listing 5.10: Code Block 5.1: DNS Event Trace Transformed Import

```
1 import pandas as pd
2 import numpy as np
3 from datetime import datetime, timezone
4 from datascience import *
5 import matplotlib.pyplot as plots
6
7 now = datetime.now()
8
9 #Standard Data Science Inline Graph Usage by DeNero et al. (2020)(pp.4)
```

```

10 %matplotlib inline
11 plots.style.use('fivethirtyeight')
12
13 #Pang DNS Failure Trace Data Set
14 traceds = pd.read_csv('event_trace.csv')
15
16 #Analysis Table Data Set for Full Event Trace Population
17 eventds = Table().with_columns('EventID',traceds.event_id)
18 eventds = eventds.with_columns('NodeName',traceds.node_name)
19 eventds = eventds.with_columns('EventStart',traceds.event_start_time)
20 eventds = eventds.with_columns('EventStartE',traceds.event_start_time)
21 eventds = eventds.with_columns('EventStop',traceds.event_stop_time)
22 eventds = eventds.with_columns('EventStopE',traceds.event_stop_time)
23 eventds = eventds.with_columns('EventType',traceds.event_type)
24 eventds = eventds.with_columns('EventTimeDiff', traceds['event_stop_time'] - traceds['event_start_time'])
25
26 eventds['EventStart'] = pd.to_datetime(eventds['EventStart'],unit='s')
27 eventds['EventStop'] = pd.to_datetime(eventds['EventStop'],unit='s')
28 eventds

```

To cluster IP-addressed nodes into an initial bin set, a custom function defines 100-interaction cluster sets.

Listing 5.11: Code Block 5.2-DNS Event Trace Clusters

```

1 # Define Clusters for Availability-Non Availability
2 def setValueCluster(SumVal):
3     if SumVal <= 100:
4         return '0-100'
5     elif SumVal <= 200:
6         return '101-200'
7     elif SumVal <= 300:
8         return '201-300'
9     elif SumVal <= 400:
10        return '301-400'
11    elif SumVal <= 500:
12        return '401-500'
13    elif SumVal <= 600:
14        return '501-600'
15
16 # Define and Present Cluster Pivot
17 DNSEventPivot = eventds.pivot("EventType",str("NodeName"))
18 DNSEventPivot = DNSEventPivot.with_column('EventSum', DNSEventPivot[1] + DNSEventPivot[2])
19 DNSEventPivot = DNSEventPivot.with_column('EventSumCategory', DNSEventPivot.apply(setValueCluster,'EventSum'))
20 print(DNSEventPivot.group("EventSumCategory"))
21
22 # Pivot Output for Pang FTA Event Trace Data Set
23 #EventSumCategory | count
24 #0-100           | 61943
25 #101-200         | 231
26 #201-300         | 15
27 #401-500         | 2
28 #501-600         | 10

```

The sum of nodes (count) that exhibit the number of interactions (EventSumCategory) indicates that the most significant number of nodes (61,943) experience from 0 to 100 interactions, while a fewer number of nodes (258) experience 101 to 600 interactions over the measurement period. None of the nodes in the data set experienced 301 to 400 interactions.

5.6.3 Significant Data Set Parameters

Pang et al. (2004) cites the structure of the DNS system as hierarchical, containing client resolvers, local name servers (LDNS), and authoritative name servers (ADNS). Client resolvers look up their local DNS server, which queries the authoritative server to perform address to hostname mapping. In the availability study by Pang et al. (2004), the primary findings are;

- “a large number of users use a small number of local DNS servers”,
- “majority of local and authoritative servers are highly available without a single observed failure”,
- “consistent availability periods last days or weeks, with recovery lasting no more than a few hours at a time”,
- “there exists a small fraction of single 9, or less, availability metrics”, and
- “local name servers are more variable in configuration and exhibit more activity on a few number of servers”, and
- “a slight positive correlation exists between usage of the node and availability for local and authoritative servers.”

The significant parameters extracted from the data set show that a small number of nodes handle the highest percentile of state switching in the measurement period. The *Most Active Nodes* code block below summarises the distinct `NodeName` instance into the combined count of total `available` and `unavailable` states over the measurement period.

Listing 5.12: Code Block 5.3-DNS Event Trace Most Active Nodes

```
1 # Display the top state switching nodes in the FTA data set
2
3 DNSMostActiveNodeSummary = Table().with_columns('NodeName', \
4         eventds['NodeName'])
5 DNSMostActiveNodeSummary = DNSMostActiveNodeSummary.with_columns\
6         ('EventType', eventds['EventType'])
7 DNSMostActiveNodeSummary.select(['NodeName']).group('NodeName').sort\
8         ('count', descending = True)
```

A descending sort on the number of combined `{1:Availability}` and `{0:Unavailability}` states indicate that there are nodes that have a high percentage of event switching interaction in the DNS event data set. This code block reiterates the findings by Pang et al. (2004). The code block result indicates that specific nodes display the highest grouping of switching behaviour in the FTA data set. For example, Node 0.0.77.231 exhibits the highest `{0:Unavailability}` and `{1:Availability}` state switching number in the data set.

Listing 5.13: Code Block 5.4-DNS Event Trace Pivots

```

1 #Event Trace Pivot for State Switching ** Use print line to validate EventSum split for each NodeName
2 #print(DNSEventPivot.sort('EventSum',descending = True))
3
4 #NodeName      | 0      | 1      | EventSum | EventSumCategory
5 #0.0.77.231    | 274    | 275    | 549      | 501-600
6 #0.0.80.155    | 273    | 274    | 547      | 501-600
7 #0.0.171.87    | 272    | 273    | 545      | 501-600
8 #0.0.63.37     | 266    | 267    | 533      | 501-600
9 #0.0.60.251    | 264    | 265    | 529      | 501-600
10 #0.0.114.94    | 263    | 264    | 527      | 501-600
11 #... (62195 rows omitted)
12
13 #Output FTA Arrays to reusable Comma Separated Value files (CSVs)
14 eventds.to_csv('DNSEventTraceTransformed.csv')
15 DNSEventPivot.group("EventSumCategory").to_csv('DNSEventClusterSummary.csv')
16 DNSMostActiveNodeSummary.select(['NodeName']).group('NodeName').sort('count',descending = True).to_csv('
    DNSMostActiveNodeSummary.csv')
17 DNSEventPivot.sort('EventSum',descending = True).to_csv('DNSMostActiveNodeDetail.csv')

```

The significant configuration parameters that facilitate a reduced data set are; **Network-Failure** and **DynamicIPAddress**, as seen in Pang et al. (2004). The code blocks *DNS Event Trace Most Active Nodes*, and *DNS Event Trace Pivot* represent a reflection of a point in time without further filtering of the data set. In Pang et al. (2004), the *NetworkFailure* parameter tested for failures close to the node from which data ping requests initiate. The node networks exhibit a clustered impact and thus supported the exclusion from the primary measurement data set. Pang removed measurement periods where many nodes indicated "correlated" failures, Pang et al. (2004). The resultant output from the LDNS data set is further constrained through a **DynamicIPAddress** parameter using a mix of name filtering, address clusters, and sequence responses.

The scale of the reachable nodes was reduced, given specific boundary conditions. These are hard system boundaries against which Pang et al. (2004) state the significant primary findings in their work. The time-bound boundary for the measurement obtained using the stats function shows that the earliest EventStart is experienced for Node 0.0.0.35 on 2004-03-18 18:24:03 with the event state ending on 2004-03-22 18:21:05. Similarly, the earliest stop is for Node 0.0.16.134 is on 2004-03-18 19:27:53 and 2004-03-18 19:52:04. Thus the minimum start event is 2004-03-18 18:24:03, and the minimum stop event is 2004-03-18 19:27:53. Similarly, the max start event is 2004-04-11 02:27:14 for Node 0.0.27.198, and the max stop event is 2004-04-11 02:27:15 for Node 0.0.91.147. The difference between the earliest start and latest stop event is the number of days between 2004-04-11 02:27:15 and 2004-03-18 18:24:03. In Epoch terms, this is the difference between 1081650435 and 1079634243, which equates to 23 days. This result can be verified using a simple difference between the Epoch dates found in the FTA data set. The standard *eventds.stats* output in Python produces the Table 5.10.

For brevity, the minimum and maximum date values are the significant parameters for the data under review.

The method of stating measurement filters condense the Pang area of study into a data

Table 5.10: FTA DNS Significant Range

Statistic	EventID	EventStart	EventStop	EventType
min	0	2004-03-18 18:24:03	2004-03-18 19:52:04	0
max	548	2004-04-11 02:27:14	2004-04-11 02:27:15	1

set that contains a suitably bounded population. The code blocks *DNS Event Trace Transformed Input* and *DNS Event Trace Clusters* represent a confirmed system boundary, with defined `{0:Unavailability}` and `{1:Availability}` states. Therefore, the FTA data set represents a point in time view of state across a set of nodes with the hierarchy of a system.

The essential data set parameters indicates that a large number of nodes switch state less frequently. These nodes are represented by conditions that align them closer to LNDS type nodes. Namely, those that require a lookup towards an ADNS node-set. The significance in the data set shows that as few as ten nodes in the entire population had the majority of available and unavailable state switching for the 23-day measurement period. Therefore, this data set provides sufficient operating conditions for the ISARC model validation. The data set conforms to; a two-tier hierarchy between address requesting nodes, a bound measurement period, and the parameters; `NetworkFailure` and `DynamicIPAddress`, as a candidate set of constraint conditions for a specific addressing process.

The population statistics for the main data set *traceds* in code block 5 - now labelled *DNSEventTraceTransformed* - shows; 223596 individually occurring `{1:Availability}` states and 161395 individually occurring `{0:Unavailability}` event states. Thus, a selection for the population set of 384991 events can be split into a 42% `{0:Unavailability}` and 58% `{1:Availability}` statistic. The population split in the code block assumes - with all things equal - that a connected state will prevail 58% of the time within each validation sample set.

In the lecture on data sampling(Lecture 4.2), Adhikari et al. (2020) cites three sampling methods to obtain a representative sample. *Deterministic sampling* determines where a fixed set of positions can be sampled from the data set, *Probability sampling* indicates that an event is selected by chance out of the population, and *Uniform Random sampling* indicates an equal chance of selection of each event from the entire population. As the fixed measurement period in *DNSEventTraceTransformed* represents points in time, a Deterministic sample can display some bias in the validation. For example, an `{0:Unavailability}` state

that continues for a more extended period lends itself to testing a more significant number of configuration options. Conversely, a shorter `{0:Unavailability}` state of a few seconds or minutes might not have the opportunity of applying that same amount of configuration changes in order to correct the `{0:Unavailability}` state. Random and uniform random sampling can be used as the population split in *DNSEventTraceTransformed* is roughly around midway. Both methods are present in the validation.

Adhikari et al. (2018)(ch.11) and Adhikari et al. (2020)(Lecture 5.3) summarise the steps for statistical model assessment. The steps for model assessment contain;

- the *definition of a statistic* from the data set. This is a measure against which the model is tested. For the ISARC, it is the runtime configuration set that comprises of parameter selection from DSM code blocks 1 to 4 that could change the fixed *DNSEventTraceTransformed* `{1:“Availability”}` and `{0:“Unavailability”}` states,
- the *simulation of the statistic* given the model’s assumptions and constraints,
- the *visualisation of the prediction* outcomes against the statistic, and
- a *claim that the model aligns or misaligns* with the statistic.

In their lecture on random sampling, Adhikari et al. (2020)(Lecture 5.2. A Model about Random Selection) produces a novel random selection code block that creates a selection on a 100 unit population between a 0.26 and 0.74 distribution split - a jury selection use case. This selection approach is further stated in Adhikari et al. (2018)(ch.11.1). The code block below modifies the random selection code indicated in Adhikari et al. (2020)(Lecture 5.2. A Model about Random Selection) and Adhikari et al. (2018)(ch.11.1), by creating a sample selection proportion from the *DNSEventTraceTransformed* population count. This randomly selected population set ensures minimal bias in the selection of events. It further aims to show how random the actual states are according to the 58/42 split that *DNSEventTraceTransformed* represents. A selected random event set from *DNSEventTraceTransformed* expands to its pre and post-event states.

Listing 5.14: Modified Sample Population Distribution

```

1 #Adapted from Adhikari et al. (2020)(Lecture 5.2. A Model about Random Selection)
2
3 connectedCount = make_array()
4 connectedSplit = make_array(0.58, 0.42)
5
6 # Total FTA Event Occurrences 384991
7 # 0.58 223596 {1:Availability} States & 0.42 161395 {0:Unavailability} States
8 eventSample = 384991
9
10 simulationTrials = 10000

```

```

11 for i in np.arange(simulationTrials):
12     sample_distribution = sample_proportions(eventSample, connectedSplit)
13     sample_count = (100*sample_distribution).item(0)
14     connectedCount = np.append(connectedCount, sample_count)
15
16 Table().with_column('Random Sample Count', connectedCount).hist()

```

The code block outputs a Python histogram that samples several available states in the *eventSample* over 10000 simulation trials. As expected, the sample indicates that the number of connected states under the code block conditions in the sample tends to represent a normal distribution. Figure 5.12, presents a null hypothesis that indicates nothing else besides a

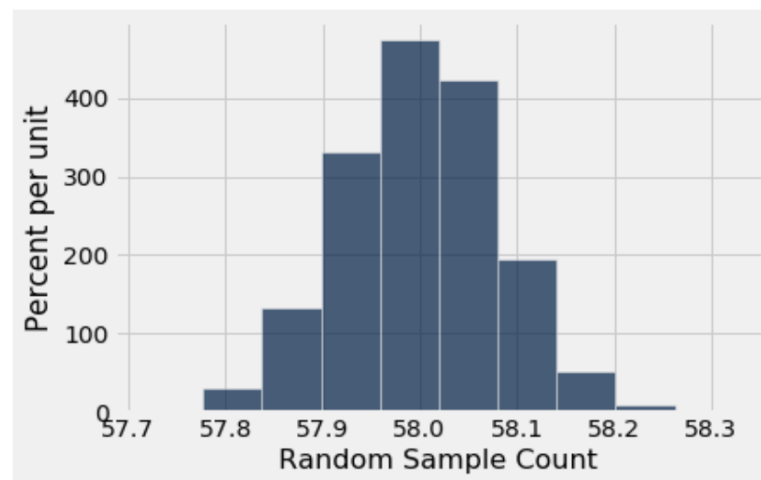


Figure 5.12: Standard Sample Population Simulation

series of random events that occurred in generating the sample data set from a statistic already represented in the FTA data set. This output does not show randomness in how the Pang et al. (2004) historical data set was generated. The code block merely shows a selection probability as a standard selection criterion given the population (384991) and the 58% **{1:Availability}** state. Adhikari et al. (2018)(ch.11.3) indicates this as a pure “chance” model. If the ISARC proof were to ignore the impacts of architecture decisions on the runtime configuration, the null hypothesis would represent that nothing besides chance is happening to the availability and unavailability states of the systems architecture, as represented in the FTA data set. The statistic of 42% **{0:Unavailability}** and 58% **{1:Availability}** is somewhat certain in that it is a direct extract from the Pang et al. (2004) historical data set. This certainty includes the assumption that nothing changes the base statistic until an action or decision impact applies to the system’s state at each measurement period in the FTA data set.

5.7 Validating ISARC Configurations

The validation of runtime configuration options accesses the outputs from Code Blocks 1 to 5 in this section. The code blocks provide the necessary flexibility to include a greater number of parameters, thereby increasing configuration options. The variables in each code block hold the necessary form and function interactions that govern the systems architecture stipulated throughout the prior chapters. The matrix decision structure native in Python mirrors the earlier arguments for system interaction and its associated hierarchies using DSMs. Extension to the DSMs and handling of additional parameters can append without introducing cascading impacts between model types. One could argue that a single code block with all required variables can produce the same validation output. Although this could be correct, it would violate primary software architecture principles such as modularity, cohesion & coupling, IEEE (1990)(pp.48). In reference to the earlier conceptual model that structures the thesis, Figure 5.13, the model validation proof is shown in Figure 5.14.

Figure 5.14 represents the implementation of the core thesis concepts of indicating con-

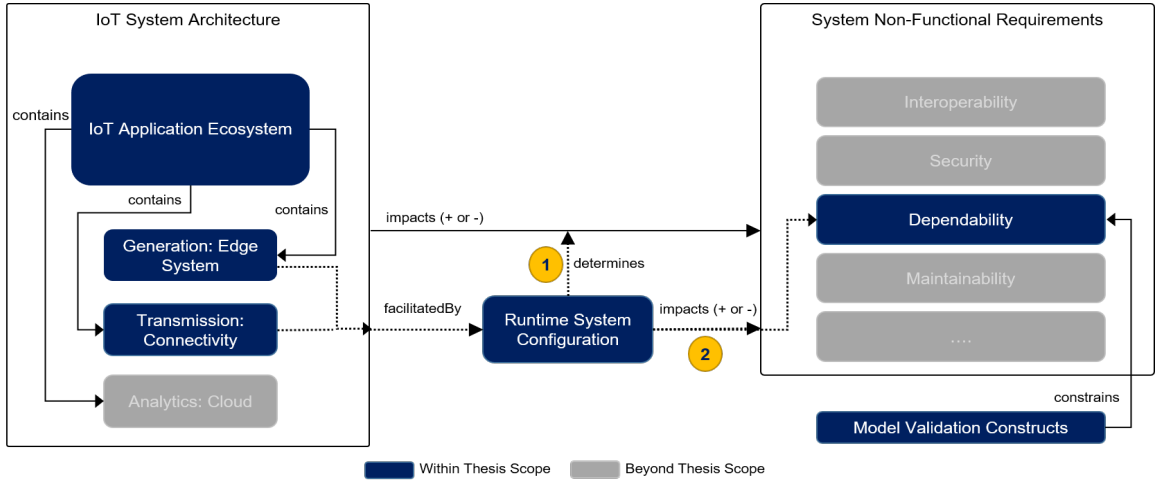


Figure 5.13: A conceptual overview of the thesis

nected edge runtime configuration options and its impact on dependability. To outline the research question responses;

1. Propose a standards-based systems architecture that outlines the runtime configuration of IoT application components found in the IoT-CE. Models 1 to 4 represent this.
2. Survey the form and function of the underlying architecture to provide an understanding of decisions that guide dependability characteristics that support state-awareness.

A Python simulation that uses *datascience* constructs proposes an implementation of (1). The model's output compares to a fixed historical data set for a multi-node DNS hierarchy, the FTA for a DNS by Pang et al. (2004).

The simulation of runtime configuration options selection thus shows how different runtime configuration parameters of the IoT system, subsystems, and components support dependability through reliable state awareness of the IoT-Ce. Additionally, the simulation of architectural options in a heterogeneous system aims to drive minimised error and control actions for the IoT-Ce ED/GW components. A non-determinism characteristic in architecture decisions appears as the primary basis for parameter selection.

Figure 5.14 extends the earlier model by referencing the code blocks that now represent each ISARC validation requirement. The simplified FTA data set handlers extend the argument for modularity in the code blocks. Code Block 5 prepares, modifies, expands, and outlines clusters of high state switching activity of the measured DNS node-set. Code Block 5 comprises of 5.1 through 5.4. The interaction set is similar to a **Front-end System Set** with multiple HTTP resource requests towards its server component. This inbound resource request references in earlier flows indicating non-blocking resource requests in Figure 3.7.

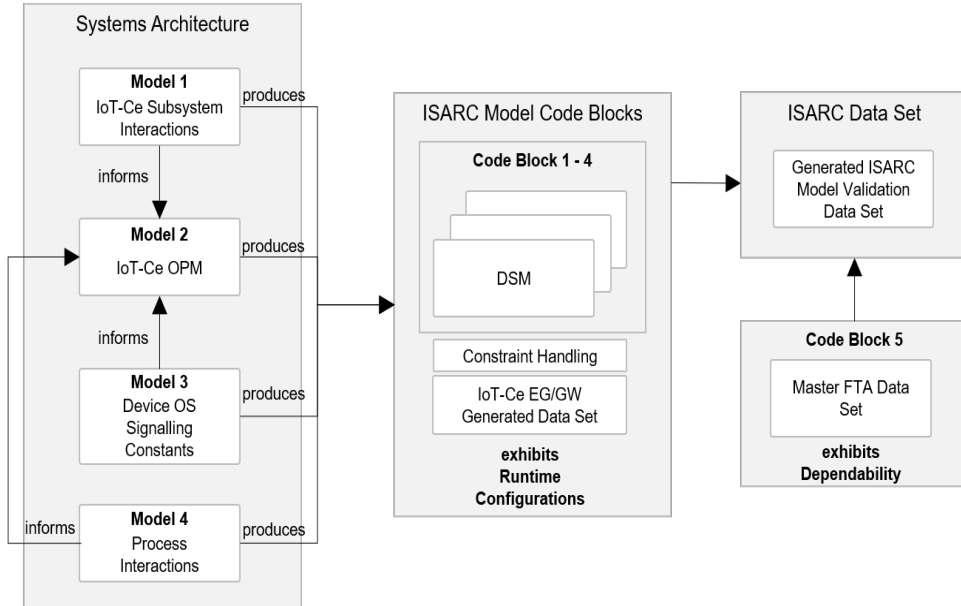


Figure 5.14: System Configuration Decision Model to Code Summary

Table 5.11 lists the input and output for all structural decisions as well the data sets that represents configuration variation testing. The generation of multiple data samples represents the suggested effectiveness of configuration options. In Python, the *datascience* library

Table 5.11: Codified ISARC Implementation and FTA Import Summary

Code Blocks (CB) for Python Variable Transformation
(CB1) dmParam1 -> decisionModel1 (9 parameters)
(CB2) dmParam2 -> decisionModel2 (26 parameters)
(CB3) dmParam3 -> decisionModel3 (36 parameters)
(CB4) dmParam4 -> decisionModel4 (6 parameters)
(CB5) FTA Trace Data Set:event_trace.csv -> DNSEventTraceTransformed:eventds
(CB5) Transformed Python Data Set:eventds -> Event Cluster Summary:DNSEventPivot
(CB5) Transformed Python Data Set:eventds -> Most Active Nodes:DNSMostActiveNodeSummary
(CB5) Event Clusters:DNSEventPivot -> Most Active Node Detail:DNSMostActiveNodeDetail

by DeNero et al. (2020), built for the online data science course material by the University of Berkeley, Adhikari et al. (2020), contains functions for the analysis of a population's statistics and inferences about those statistics.

The primary data sets for the validations initialise via the `_GlobalVariableHandler` code block. This script ensures that a measurable set of parameters represent a point in time function and form. Inferential data sets represent the state changes against this baseline. The *datascience* library by DeNero et al. (2020) generate these inferential datasets. Finally, the fixed FTA `{0:Unavailability}` and `{1:Availability}` states per node creates the mastered state blueprint.

Listing 5.15: `_GlobalVariableHandler`

```

1 #Initialise Arrays from Decision Model 1 to 4
2 #Code Block: _GlobalVariableHandler
3 import numpy as np
4 from datascience import *
5 import pandas as pd
6 import matplotlib
7 matplotlib.use('Agg')
8
9 from datascience import Table
10 %matplotlib inline
11 import matplotlib.pyplot as plt
12 plt.style.use('fivethirtyeight')
13
14 #Load Decision Models into runtime Notebook
15 decisionModel1 = np.load('DecisionModel1Property.npy')
16 decisionModel2 = np.load('DecisionModel2Property.npy')
17 decisionModel3 = np.load('DecisionModel3Property.npy')
18 decisionModel4 = np.load('DecisionModel4Property.npy')
19
20 #FTA Data Science Table Imports
21 DNSEventTraceTransformed = Table.read_table('DNSEventTraceTransformed.csv')
22 DNSEventClusterSummary = Table.read_table('DNSEventClusterSummary.csv')
23 DNSMostActiveNodeSummary = Table.read_table('DNSMostActiveNodeSummary.csv')
24 DNSMostActiveNodeDetail = Table.read_table('DNSMostActiveNodeDetail.csv')
25
26 #Initialise Result Set Data Structure
27 ISARCResultSetASimulation = Table(make_array('SimulationTime', 'FailedConnectionCount', '
    FailedConnectionPercentage', 'SuccessConnectionCount', 'SuccessConnectionPercentage', 'ConnectionMatch', '
    EDGWSamplePopulation', 'FTASamplePopulation'))
28 ISARCResultSetBSimulation = Table(make_array('SimulationTime', 'SimulationTimestep', 'ConnectionPossibilityAlt', '
    count', 'ConnectionMatch', 'PercentageSplit'))
29 ISARCResultSetBSimulationComparison = Table(make_array('SimulationTime', 'ISARCFCC', 'ISARCFCP', 'ISARCSCC', '
    ISARCSCP', 'ISARCPopulationSize', 'FTAFCF', 'FTAFCP', 'FTASCC', 'FTASCP', 'FTAPopulationSize', 'FCPMargin',

```

```

30 SCPMargin','DecisionSensitivity','ISARCConfCluster','AdjustedISARCMargin'))
ISARCResultSetBSimulationComparison_AppliedControl = Table(make_array("SimulationTime","EventStart","
FTASampleFC","FTASampleSC","FTASimulationSample","SampledFTAFCP","SampledFTASCP","FTA Unavailability
Actual","FTA Availability Actual","ISARC Unavailability Controlled","ISARC Availability Controlled"))

```

The data sets initialised in the `_GlobalVariableHandler` code block enable a single comparative baseline for the ongoing test of a client to local to remote server addressing pattern. The data set content appear as;

1. a *DNSEventTraceTransformed datascience* table which contains a series of fixed `{0:Unavailability}` and `{1:Availability}` states,
2. a *DNSEventClusterSummary datascience* table which indicates clusters of state switching behaviour,
3. a *DNSMostActiveNodeSummary datascience* table which indicates the most active state switching nodes, and
4. a *DNSMostActiveNodeDetail datascience* table, which indicates the most active node's state switching behaviour.

Furthermore the `_GlobalVariableHandler` code block delivers;

1. an *ISARCResultSetASimulation datascience* table that holds the results array of the first simulation sets,
2. an *ISARCResultSetBSimulation datascience* table that holds the results array of the second simulation sets,
3. an *ISARCResultSetBSimulationComparison datascience* table that holds comparative statistics, and
4. an *ISARCResultSetBSimulationComparison_AppliedControl datascience* table that holds comparative statistics with applied PI control.

5.8 Sampled Data Set for Fixed FTA Measurements

The variation in sample set indicated in the lectures on sampling in Adhikari et al. (2020) show that the *DNSEventTraceTransformed* data set can split into fixed or random sampling sizes. For example, a fixed sample could contain lower frequency interaction addressing components for `Front-end System Set` instances. A 0-100 interaction set will contain variable `Front-end System Set` connection attempts with a lower address resolution success rate if nodes in the 101-600 interaction set are in state `{0:Unavailability}`. The larger number of

Front-end System Set instances in the 0-100 interaction set still adheres to random placement as per Bera et al. (2016). Additionally, a 101-600 interaction set will attach **Front-end System Set** instances with higher switching behaviour. For brevity, a larger number of nodes exhibit fewer interactions when compared to a concentrated cluster of specific nodes. Hence, it exhibits lower addressing success for **Front-end System Set** instances. A fewer number of nodes exhibit a larger number of interactions.

Due to this characteristic, several sample variations from the primary *DNSEventTraceTransformed* data set represent **Front-end System Set** interactions. Simulation code that enables a repeatable extract of sample data set variations appears in the *_GlobalEventSampling* code block. Three different sample approaches appear for clarity; however, the choice of sampling is to conform to a uniform random sample for uniquely sampled events.

Listing 5.16: *_GlobalEventSampling* Summary

```

1 #Comment Code by adding or removing Hash Symbol for Specific Sample Base Types (BT)
2 #Code Block: _GlobalEventSampling
3
4 #An example of a Deterministic Sample for record locators 10,14 and 19.
5 #Input: DNSEventTraceTransformed
6
7 DNSEventTraceTransformed.take[10,14,19]
8
9 #An example of a Deterministic Sample for nodes with .1 Tuple in the Node Address.
10 DNSEventTraceTransformed.where("NodeName", are.containing('.1'))
11
12 #An example of a Deterministic Sampling with Random Start
13 sampleStart = np.random.choice(np.arange(10))
14 DNSEventTraceTransformed.take(np.arange(sampleStart, 10))
15
16 #An example of a Uniform Random Sample with Replacement. Showing Possible Duplicate Sample Events.
17 DNSEventTraceTransformed.sample(100)
18 DNSEventTraceTransformed.sample(10000).group("NodeName")
19
20 #An example of a Uniform Random Sample without replacement. Showing Uniquely Sample Events.
21 DNSEventTraceTransformed.sample(1000, with_replacement=False).group("NodeName")
22 DNSEventTraceTransformed.sample(5, with_replacement=False)

```

The codified representation of all required decision models (CB1 through 4), data samples (CB5), and visualisations represent the core argument that the hierarchical systems architecture for the connected edge contains a multi-leveled set of form and function constructs. The following section suggests the proof of considering a well-structured parameter-based form and function-based method for IoT-Ce. Through this architectural validation approach, the thesis stipulates that it is plausible to validate variations in IoT-Ce configuration quantitatively by using a structured systems architecture based on OPM.

5.9 Sampled Data Set for Fixed Measurements using Applied ISARC Variations

In the *DNSEventClusterSummary* table, the node switching behaviour in the FTA data set shows that a larger number of nodes (count) yield a smaller number of interactions (EventSumCategory). Therefore, the ISARC model proof presents the 0-100 interaction switching cluster as the variable local addressing requirement for **Front-end System Set** nodes, representing variability in LDNS architectures. This view suggests that **Front-end System Set** instances increase to a limit within a physical spatial location. The remainder of the nodes in clusters 101-200 to 501-600, classify as more active addressing nodes due to the smaller number of nodes experiencing higher volumes of state switching behaviour. In the structure of a standard DNS, client resolvers query local name servers that, in turn, query authoritative servers to perform an address-to-name mapping. The client resolver classifies as **Front-end System Set** instances, whereas the LDNS and ADNS classify as addressing nodes.

Listing 5.17: DNSEventClusterSummary Validation

```
1 #Code Block: _MasterDataSetValidation
2
3 #DNSEventClusterSummary Output Validation
4 #EventSumCategory | count
5 #0-100            | 61943
6 #101-200          | 231
7 #201-300          | 15
8 #401-500          | 2
9 #501-600          | 10
10
11 #DNSEventClusterSummary validation against DNSEventTraceTransformed
12 MasterReferenceValidation = DNSEventTraceTransformed.join('NodeName', DNSMostActiveNodeDetail).groups(['
    EventSumCategory', 'NodeName'])
13 MasterReferenceValidation
14
15 #EventSumCategory | NodeName | count
16 #0-100            | 0.0.0.1   | 39
17 #0-100            | 0.0.0.10  | 3
18 #0-100            | 0.0.0.100 | 1
19 #0-100            | 0.0.0.101 | 33
20 #0-100            | 0.0.0.102 | 1
21 #0-100            | 0.0.0.103 | 1
22 #0-100            | 0.0.0.104 | 5
23 #0-100            | 0.0.0.105 | 19
24 #0-100            | 0.0.0.106 | 3
25 #0-100            | 0.0.0.107 | 5
26 #... (62191 rows omitted)
27
28 #MasterReferenceValidation validated against DNSEventClusterSummary
29 MasterReferenceValidation.group('EventSumCategory')
30
31 #EventSumCategory | count
32 #0-100            | 61943
33 #101-200          | 231
34 #201-300          | 15
35 #401-500          | 2
```



```

36 | #501-600          | 10
37 |
38 | #No Instances of 301 to 400 Event Interactions Found

```

The `_MasterDataSetValidation` code block confirms that the aggregate transformations completed in *DNSEventTraceTransformed* and *DNSMostActiveNodeDetail* do not yield anything outside of the original data sets. This validation exists to indicate that the thesis does not change the main *event_trace.csv* data set by adding or modifying record values that occur in the original measurement set. This validation is important as the output result sets have to represent a generated data set devoid of any links apart from those necessary to compare the generated data set to the original historical FTA data set. The thesis does not, under any circumstances, generate the output using record-level information from the original data set. The proof of this is evident in the following section.

5.9.1 ISARC Data Set Generator

The generated ISARC data set generator in code block `_GlobalDataSetGenerator` generates *Front-end System Set* connection attempts to the LDNS nodes population in the *DNSEventTraceTransformed* data set.

Listing 5.18: `_GlobalDataSetGenerator`

```

1 | #IoT Systems Architecture for Runtime Configuration (ISARC) Data Generation
2 | #Code Block: _GlobalDataSetGenerator
3 | #Output: Front-end System Set (FE-SS) DNS Attempt Events
4 | #Output: DNSConnectionEventTraceGenerated
5 |
6 | isarcDSTmp = Table(make_array('RowID', 'EventID', 'NodeName', 'CandidateDNS', 'EventStartE', 'EventStopE', '
   |   EventAttempt'))
7 |
8 | #Number of Rows in Generated Data Set
9 | rowID = 100000
10 | #All Active Nodes. No Inactive States
11 | state = [1]
12 | #Upper Limit for Front-end System Set Node Occurrence
13 | maxNodes = 50000
14 | #Variable Connection Termination Period
15 | connectionTerminated = [1,5,10,20,30]
16 |
17 | for _ in np.arange(rowID):
18 |     addressSeg3 = np.random.choice(np.arange(249))
19 |     addressSeg4 = np.random.choice(np.arange(255))
20 |     address = "FE-SS" + '-' + str(np.random.choice(np.arange(maxNodes)))
21 |     CandidateDNS = str("0") + '.' + str("0") + '.' + str(addressSeg3) + '.' + str(addressSeg4)
22 |     eventStart = np.random.choice(DNSEventTraceTransformed['EventStartE'])
23 |     eventEnd = eventStart + np.random.choice(connectionTerminated)
24 |     eventAttempt = np.random.choice(state)
25 |     isarcDSTmp.append([_, '1', address, CandidateDNS, eventStart, eventEnd, eventAttempt])
26 |
27 | isarcDSTable['EventStart'] = pd.to_datetime(isarcDSTable['EventStartE'],unit='s')
28 | isarcDSTable['EventStop'] = pd.to_datetime(isarcDSTable['EventStopE'],unit='s')
29 |
30 | #Generate Customer ISARC Measurement Event Data Set
31 | DNSConnectionEventTraceGenerated = isarcDSTable.select([
32 |     'RowID',

```

```

33     'EventID',
34     'NodeName',
35     'CandidateDNS',
36     'EventStart',
37     'EventStop',
38     'EventAttempt'
39 ]
40 DNSConnectionEventTraceGenerated
41 DNSConnectionEventTraceGenerated.to_csv(".\isarcSimulations\DNSConnectionEventTraceGenerated_" + now.strftime(
    "%m%d%Y_%H%M%S") + ".csv")

```

The ISARC generated dataset contains start and end date markers that create a fixed measurement period between the earliest start and latest stop dates for generated events. Line 23 and 24 in the `_GlobalDataSetGenerator` produce a random event start date that coincides with the range of dates in *DNSEventTraceTransformed* EventStartE field. This constraint ensures that the tests execute within the bound FTA measurement period from 2004-03-18 18:24:03 to 2004-04-11 02:27:14. Additionally, it creates the measurement boundary that creates an absolute measurement when applied to the *DNSEventTraceTransformed* data set. Furthermore, each **Front-end System Set** instance has a fixed connection terminating constraint array containing [1,5,10,20,30] seconds. To recall, the [IoT Data Transmitting SD1.1](#) in Chapter 4 contained an optional constraint that indicated **Device Data Sending** triggers **Connecting Terminating** when it lasts more than 5 seconds. The address segments in the code block `_GlobalDataSetGenerator` also create the appropriate **Front-end System Set** nodes using a segmented generator. The finding is that on fewer generated rows, the chance that a random **Front-end System Set** connecting to a node in the 0-100 cluster segment is equally probable to it appearing in the 101-600 cluster. This finding is present without redefined decision model interactions or constraints. Lines 14, 20, and 25 produce the unique node connection event generation process. The Candidate DNS IP address segment structure has the range;

1. 0 to 0 for address segment 1,
2. 0 to 0 for address segment 2,
3. 0 to 249 for address segment 3, and
4. 0 to 255 for address segment 4.

As an ascending sort on the node name in the *DNSEventTraceTransformed* data set, 0.0.0.1 appears as the first node. The lower end nodes and the final nodes for the third address segment max is 0.0.249.162, with the node that contains the fourth address segment max is 0.0.248.255. Therefore, address segments 3 and 4 reflect the *DNSEventTraceTransformed*

node population but do not create a fixed relationship link between the historical and generated data sets. The Candidate DNS IP in the ISARC generated data set merely enables relational context.

5.9.2 Decision Model Parameter Handler

A static decision baseline for ISARC Decision Model 1 to ISARC Decision Model 4 is set using the known and perceived interactions stated in the DSMs in Chapter 4, section 4.6. Decision Model 1 states that **M1.C1.1:Communication** interacts as an input to **M1.C1.2:HardwareAbstraction**, and vice versa. This interaction means that decisions relating to **M1.C1.2:HardwareAbstraction** selection has an input impact with **M1.C1.1:Communication**, and due to co-design requirements **M1.C1.1:Communication** requirements conversely impact **M1.C1.2:HardwareAbstraction** decisions. Within the system boundary, systemic physical elements such as power and layout specification could impact hardware selection, thus impacting **M1.C1.1:Communication**. However, a strict system boundary at the top-level node of the **M1.C1.2:HardwareAbstraction** hierarchy protects cascading decision impacts. Due to the nature of physical hardware, the specification can vary, and thus a decision can be made within the hierarchy by extending the hierarchy tree. This modelling concept appears through in-zooming and unfolding systems capability in OPM ISO-19450. A structural hierarchy that uses all decision models with defined probabilities as a selection priority appears in the `_ISARCDecisionParameterSelection` code block.

The `isarcConfigTable` in the `_ISARCDecisionParameterSelection` output produces a single configuration table containing all decision parameters in the required decision space, their model location, and a decision sensitivity of 0.001 for all decision parameters in the specified decision space.

Listing 5.19: ISARC Decision Model Primary Data Set

```

1 #Base Data Sets for Fixed Structural Interactions and Variable Parameter Selections
2 #Code Block: _ISARCDecisionParameterSelection
3 #Output: Converged ISARC Configuration Table
4
5 #Minimum Selection Priority for (DM1,DM2,DM3,DM4)
6 minSelectionPriority = make_array(0, 0.001, 0.001, 0.001, 0.001)
7 #Optional Maximum Selection Priority for (DM1,DM2,DM3,DM4)
8 maxSelectionPriority = make_array(0, 0.999, 0.999, 0.999, 0.999)
9
10 selectionPriority = minSelectionPriority
11 isarcConfigurationTmp = Table(make_array('SourceID', 'ArchRef', 'ISARCDecisionLocator', 'ISARC_StructuralSValue',
12                                          'ISARC_DetectedSValue'))
13 #Apply Minimum Selection Priority against all Parameters)
14 for _ in np.arange(len(decisionModel1)):
15     isarcConfigTable = isarcConfigurationTmp.append(['DSM1', decisionModel1[_][0], decisionModel1[_][1],
16                                                     selectionPriority[1],0])
17 for _ in np.arange(len(decisionModel2)):

```

```

17     isarcConfigTable = isarcConfigurationTmp.append(['DSM2', decisionModel2[_][0], decisionModel2[_][3],
18         selectionPriority[2],0])
19 for _ in np.arange(len(decisionModel3)):
20     isarcConfigTable = isarcConfigurationTmp.append(['DSM3', decisionModel3[_][0], decisionModel3[_][2],
21         selectionPriority[3],0])
22 for _ in np.arange(len(decisionModel4)):
23     isarcConfigTable = isarcConfigurationTmp.append(['DSM4', decisionModel4[_][0], decisionModel4[_][1],
24         selectionPriority[4],0])
25
26 RowKey = np.arange(isarcConfigTable.num_rows)
27 isarcConfigTable = isarcConfigTable.with_column('RowKey',RowKey)
28 isarcConfigTable.show()
29
30 #Output: Converged ISARC Configuration Table
31 #SourceID | ArchRef | ISARCDecisionLocator | ISARC_StructuralSValue | ISARC_DetectedSValue | RowKey
32 #DSM1 | M1.C1.1 | Communication | 0.001 | 0 | 0
33 #DSM1 | M1.C1.2 | HardwareAbstraction | 0.001 | 0 | 1
34 #DSM1 | M1.C1.3 | OperatingSystem | 0.001 | 0 | 2
35 #DSM1 | M1.C1.4 | RemoteManagement | 0.001 | 0 | 3
36 #DSM1 | M1.C2.1 | DataManagementMessaging | 0.001 | 0 | 4
37 #DSM1 | M1.C2.2 | Communication | 0.001 | 0 | 5
38 #DSM1 | M1.C2.3 | ApplicationRuntime | 0.001 | 0 | 6
39 #DSM1 | M1.C2.4 | OperatingSystem | 0.001 | 0 | 7
40 #DSM1 | M1.C2.5 | RemoteManagement | 0.001 | 0 | 8
41 #DSM2 | M2.1 | IoT Edge System Value Delivering | 0.001 | 0 | 9
42 #... (67 rows omitted)

```

The *_ISARCDecisionParameterSelection* applies a consistent parameter selection value (ISARC_StructuralSValue). The parameter selection field lends itself to primary classes of impact such as High(H), Medium(M), or Low(L). Alternatively, the DSM intersection is also amenable to impact probability. The (H) impact probability could contain a narrow probability range that tends to 1.000 - such as 0.991 to 0.999. An (L) impact probability that tends towards 0.000 - such as 0.001 to 0.004 - would have a lower decision impact probability. A wide range such as 0.001 to 0.999 leaves many selection options available and does not serve a specific deterministic purpose when making an architecture decision. A suitable decision space is a known principle heuristic presented in Crawley et al. (2016). The decision space that contains clusters of impacts in the *isarcConfigTable* table outlines a relationship between decision parameters (ISARCDecisionLocator). With all *ISARC_StructuralSValue* instances set to 0.001, it indicates that everything has an equal selection probability. By adjusting individual decisions, they naturally form part of an IoT-Ce configuration set. The input to the decision model uses structured IoT-Ce parameters to create an architectural space within which simulations and state impact arguments form.

The following code block indicates the parameter modifier selection code block that enables the shifting and clustering of ISARC decisions into configuration variations. The setting of decision clusters enables the appropriate hierarchy value and impact sensitivity given different IoT-Ce configuration sets.

The modification below is a manual adjustment that provides support for dynamic *ISARC_StructuralSValue* handling. Additionally, *ISARC_DetectedSValue* enables expansion to provide comparative parameter distance values from an internal array or external source of data.

Listing 5.20: ISARC Decision Model Manual Setting of Selection Modifier

```

1 #Base Data Sets for Fixed Structural Interactions and Variable Parameter Selections
2 #Code Block: _ISARCDecisionParameterSelectionModify
3 #Output: ISARC Configuration Table with Examples of Manually Updated Selection Modifiers
4
5 #Minimum Selection Priority for (dm1,dm2,dm3,dm4)
6 minSelectionPriority = make_array(0, 0.001, 0.001, 0.001, 0.001)
7
8 #Maximum Selection Priority for (dm1,dm2,dm3,dm4)
9 maxSelectionPriority = make_array(0, 0.999, 0.999, 0.999, 0.999)
10 selectionPriority = minSelectionPriority
11
12 #Selection Modifier Start using RowKey Value (Option 1 using ISARC Paramter Location)
13 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][0] = 0.999
14 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][5] = 0.999
15
16 #Selection Modifier Start using ISARC Decision Locator Value (Option 2 using Paramter ISARC Name)
17 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][isarcConfigTable.where("
18     ISARCDecisionLocator", are.equal_to("Created"))['RowKey']] = 0.999
19 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][isarcConfigTable.where("
20     ISARCDecisionLocator", are.equal_to("Running"))['RowKey']] = 0.999
21 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][isarcConfigTable.where("
22     ISARCDecisionLocator", are.equal_to("Completed"))['RowKey']] = 0.999
23
24 #Output: ISARC Configuration Table with Examples of Manually Updated Selection Modifiers
25 #SourceID | ArchRef | ISARCDecisionLocator | ISARC_StructuralSValue | ISARC_DetectedSValue | RowKey
26 #DSM1 | M1.C1.1 | Communication | 0.999 | 0 | 0
27 #DSM1 | M1.C2.2 | Communication | 0.999 | 0 | 5
28 #DSM4 | M4.1-A | Created | 0.999 | 0 | 71
29 #DSM4 | M4.2.B | Running | 0.999 | 0 | 72
30 #DSM4 | M4.4.D | Completed | 0.999 | 0 | 74

```

The output of 100000 simulation trials containing ten random selections from the 77-ISARC parameter list, represents a limited configuration set for the [Front-end System Set](#).

Even in the case where [Communications](#), and [Front-end System Set](#) process cycle containing [created](#), [running](#) and [completed](#) states are important, this configuration selection yields a stepped average, from a pure average selection value point of view.

Listing 5.21: ISARC Decision Model Simulation Output

```

1 #Base Data Sets for Fixed Structural Interactions and Variable Parameter Selections
2 #Code Block: _ISARCDecisionParameterSelection
3 #Output: ISARC Simulated Configuration Table
4 #Output: AverageParameterDecisionSelectionProbability.html
5
6 minselectionCount = []
7 maxselectionCount = []
8 selectionCount = []

```

```

9 selectionPopulation = 10
10 simulationTrials = 100000
11
12 for i in np.arange(simulationTrials):
13     sampledSelectionAverage = np.average(isarcConfigTable.sample(selectionPopulation,with_replacement=False)['
14         ISARC_StructuralSValue'].astype(np.float))
15     selectionCount = np.append(selectionCount,sampledSelectionAverage)
16 Table().with_column('Averaged Parameter Selection Value',selectionCount).hist()

```

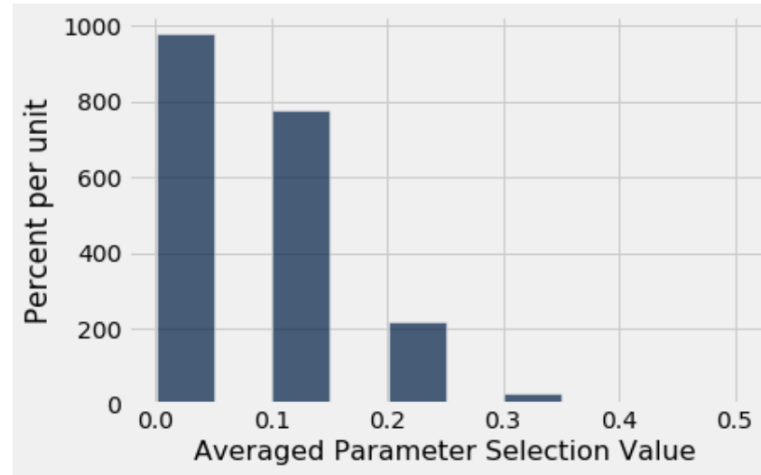


Figure 5.15: Decisions Based on Structural Integrity of System

Figure 5.15 presents the outcome of a 100000 cycle simulation set containing a random sample of ten individual parameters from the ISARC parameter configuration table. The result indicates that the consideration for lower decision sensitivity selection on the parameter yields a more significant number of non-action considerations. This approach yields a no better result than a pure non-deterministic view of architecture configuration variation handling.

To indicate the reference back to the decision model DSMs in Chapter 4, the intersection of each row and each column in the DSM indicates a sensitivity for each interaction decision. As indicated, each row supports the associated column using the IC format; namely, the individual row and column intersection indicate an input from the row instance towards the column instance. The IC intersection holds a value that ranges between the upper limit of sensitivity (0.999) and the lower limit (0.001). As the value tends towards 0.999, it suggests that a more significant impact exists between interacting components. A higher selection probability assignment assigns greater confidence that the parameter/decision interaction is necessary in this interaction case. In the case of the connected edge, a [Front-end System Set](#) applies a single boundary for both edge and IoT gateway components, and thus M1.C1.1 and M1.C2.2 are tightly coupled. Therefore, a decision relating to M1.C1.1 and M1.C2.2 would

also require a [M2.17:Communication](#) Capability. Similarly, [M2.11:Transmitting](#) would have a higher sensitivity coupled to [M2.12:Transmission Attempt](#), and [M2.25:Device Data Sending](#).

The ISARC structural value - how the system interacts - is marked at 0.999 for interactions in Decision Model 1. This value relays communication dependency within [Front-end System Set](#). One can suggest a change of the individual StructuralSValue such that input and output probabilities vary away 0.999. Additionally, the earlier pre-existence view on [Addressing](#) serves as a lower StructuralSValue but is included due to [Addressing](#) that precedes [Identifying](#) which precedes [Communicating](#) in the systems hierarchy. This change will aim to decide whether internal [Front-end System Set communication](#) is core to the parameter selection within a single configuration. Values that tend towards 0.001 represents a lower decision sensitivity.

In code form, the increase and decreases modification appear as;

Listing 5.22: Front-end System Set Communication Interaction Sensitivity Value

```
1 isarcConfigTable.where('ISARCDecisionLocator', are.containing('Communication'))
2
3 #Output
4 #SourceID|Param.Ref |ISARCDecisionLocator |StructuralSValue |DetectedSValue
5 #DSM1 |M1.C1.1 |Communication |0.999 |0
6 #DSM1 |M1.C2.2 |Communication |0.999 |0
7 #DSM2 |M2.17 |Communication Capability |0.001 |0
```

Listing 5.23: Front-end System Set Communication Interaction Sensitivity Value Modified

```
1 isarcConfigTable[isarcConfigTable.column_index('ISARC_StructuralSValue')][isarcConfigTable.where("
  ISARCDecisionLocator", are.containing("Communication"))['RowKey']] = 0.999
2
3 isarcConfigTable.where('ISARCDecisionLocator', are.containing('Communication'))
4
5 #Modified Output
6 #SourceID|Param.Ref |ISARCDecisionLocator |StructuralSValue |DetectedSValue
7 #DSM1 |M1.C1.1 |Communication |0.999 |0
8 #DSM1 |M1.C2.2 |Communication |0.999 |0
9 #DSM2 |M2.17 |Communication Capability |0.999 |0
```

5.10 Results Summary

The results data set compares the generated ISARC data set against the FTA data set. By sampling a total of 10000, 20000, 50000, and finally, the total population of the FTA data set - 384991 records - uniformly at random, the sampled output creates a matched [Front-end System Set](#)-LNDS node-set. This matched node-set delineates the ISARC generated data set and the static FTA data set. The important notation of the ISARC generated

data set code block in section 5.9.1 is that no hard link exists between the generation of the ISARC Data Set Generator and the historical FTA data set. Such a link would exist if the generation patterns used the measurement split of 42% `{0:Unavailability}` and 58% `{1:Availability}`. This relation is not the case in the *DNSConnectionEventTraceGenerated* data set. In the *DNSConnectionEventTraceGenerated* data set;

- a smaller total population exists, 100000 compared to 384991,
- a random set of nodes within the DNS third and fourth IP address segment range is present, and
- a connected state for each node is present.

The results summary outlines the following approach to enable repeatability of the simulation under controlled conditions.

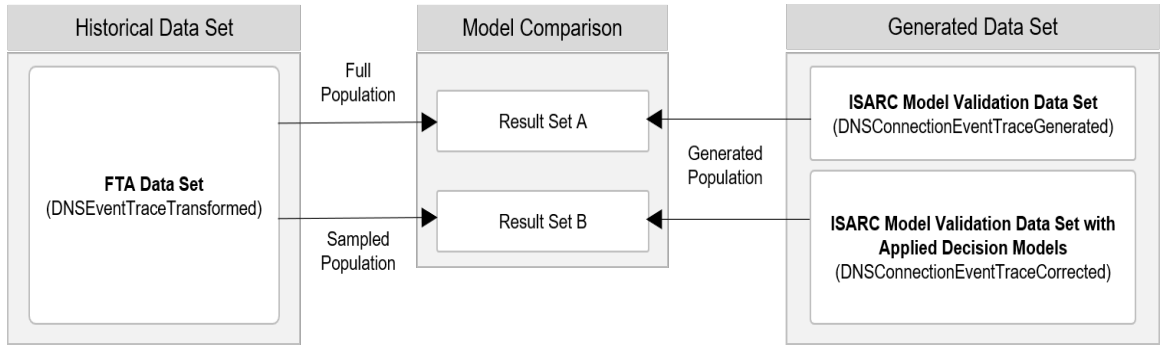


Figure 5.16: Results Summary Overview

5.10.1 Result Set A Summary

The *ISARCResultSetA* joins the generated `Front-end System Set` connection attempt dataset (*DNSConnectionEventTraceGenerated*) to the static historical data set (*DNSEventTraceTransformed*). The *DNSEventTraceTransformed* presents a point in time measurement. In contrast, the *DNSConnectionEventTraceGenerated* presents a data set that reflects a random placement, request time, and `Front-end System Set` connection attempt, which requires interaction with the associated LDNS component. The sample in the output tests the join function by stipulating a random third and fourth address segments in the candidate LDNS node. As a smoothing mechanism, the *DNSConnectionEventTraceGenerated* contains a second node tuple that assigns the node number which requires a response from the LDNS candidate node. The `_GlobalDataSetGenerator` code block indicates a `maxNode` parameter that applies the constraint of increasing nodes per assignment, up to a maximum limit, as shown in the constraints section 5.5.3.

The combination of the first, second, third, and fourth address segments in an IP address determines the nodes in class A, B, or C network, however, the addresses in the *DNSEventTraceTransformed* reflect unique node instances. Hence, with blank first and second address segment values in the *DNSEventTraceTransformed* data set, a simulation of a candidate DNS IP address value is possible. The candidate node address is unique and logically assigns to a unique LDNS entry. The ISARC simulation does not replace addressing requirements but instead builds on the uniqueness of the finding in the *DNSEventTraceTransformed*. For brevity, the Pang et al. (2004) data set indicates that a smaller number of LDNS nodes experience a higher amount of switching behaviours. Finally, given that the *DNSEventTraceTransformed* data set contains a set of fixed point in time measurements, the transformation from Pang's original source data to the *DNSEventTraceTransformed* data set does not modify or change the content of the measurement set produced by Pang. The *Results Set A Sample Output* code block is the repeatable simulation that contains several variable switches such as configurable sample sizes for sample1 (DNSConnectionEventTraceGenerated) and sample2 (DNSEventTraceTransformed) data sets. The simulation is capable of applying the following switches;

1. Execute multiple samples for resultsSetASampleSize where sample1 = 10000 and sample2 10000
2. Execute multiple samples for resultsSetASampleSize where sample1 = 20000 and sample2 20000
3. Execute multiple samples for resultsSetASampleSize where sample1 = 50000 and sample2 50000
4. Execute multiple samples for resultsSetASampleSize where sample1 = 50000 and sample2 384991

Listing 5.24: Results Set A Sample Output

```

1 #IoT Systems Architecture for Runtime Configuration Results Sample
2 #Code Block: \_ResultSetANative
3 #Output: ISARCResultSetA, ISARCResultSetASummary, and ISARCResultSetASimulation
4
5 resultsSetASampleSize = 50000
6 sample1 = DNSConnectionEventTraceGenerated.sample(resultsSetASampleSize, with_replacement=False)
7 sample2 = DNSEventTraceTransformed
8 samplesTmp = sample1.join('CandidateDNS', sample2, 'NodeName')
9
10 samplesTmp_ = samplesTmp.relabel("EventStart", "EDGWConnEventStart")
11 samplesTmp_ = samplesTmp.relabel("EventStop", "EDGWConnEventStop")
12 samplesTmp_ = samplesTmp.relabel("EventAttempt", "EDGWConnEventAttempt")
13 samplesTmp_ = samplesTmp.relabel("EventID_2", "DNSEventID")

```

```

14 samplesTmp_ = samplesTmp.relabel("EventStart_2", "DNSStateStart")
15 samplesTmp_ = samplesTmp.relabel("EventStop_2", "DNSStateStop")
16 samplesTmp_ = samplesTmp.relabel("EventType", "DNSEventState")
17 samplesTmp_ = samplesTmp.relabel("EventTimeDiff", "DNSEventStatePeriod")
18
19 ISARCRResultSetA = samplesTmp_.select(
20     "RowID",
21     "EventID",
22     "NodeName",
23     "CandidateDNS",
24     "EDGWConnEventStart",
25     "EDGWConnEventStop",
26     "EDGWConnEventAttempt",
27     "DNSEventID",
28     "DNSStateStart",
29     "DNSStateStop",
30     "DNSEventState",
31     "DNSEventStatePeriod"
32 )
33 #Code Segment: Result Set Connection Attempt Comparison Function
34 #Output: Result Set A
35 #Output: ISARCRResultSetA_
36 ISARCRResultSetA = ISARCRResultSetA.with_column(
37     "Connected", ISARCRResultSetA.apply(lambda x, y: x - y, 'DNSEventState', 'EDGWConnEventAttempt')
38 )
39 def setConnectionAttempt(connected):
40     if connected == 0:
41         return 'Success'
42     elif connected != 0:
43         return 'Failed to Connect'
44
45 ISARCRResultSetA = ISARCRResultSetA.with_column('ConnectionPossibility', ISARCRResultSetA.apply(
46     setConnectionAttempt, 'Connected'))
47 ISARCRResultSetASummary = ISARCRResultSetA.group("ConnectionPossibility")
48 ISARCRResultSetASummary = ISARCRResultSetASummary.with_column('ConnectionMatch', ISARCRResultSetA.num_rows)
49 ISARCRResultSetASummary = ISARCRResultSetASummary.with_column(
50     "PercentageSplit", ISARCRResultSetASummary.apply(lambda x, y: (x / y) * 100, 'count', 'ConnectionMatch')
51 )
52 ISARCRResultSetASimulation = ISARCRResultSetASimulation.append([
53     pd.to_datetime('now'),
54     ISARCRResultSetASummary[1][0],
55     ISARCRResultSetASummary[3][0],
56     ISARCRResultSetASummary[1][1],
57     ISARCRResultSetASummary[3][1],
58     ISARCRResultSetA.num_rows,
59     sample1.num_rows,
60     sample2.num_rows
61 ])
62 ISARCRResultSetASimulation.to_csv(".\isarcSimulations\ISARCRResultSetASimulation.csv")
63 ISARCRResultSetASimulation.num_rows

```

The output of Simulation Sample Set 1 results from the execution of 10 trials for *resultsSetASampleSize* where *DNSConnectionEventTraceGenerated* = 10000 and *DNSEventTraceTransformed* = 10000. The output of the *ISARCRResultSetA* data set thus attaches a randomly connected ED/GW, with random connection termination requirements, against a fixed LDNS availability, and unavailability, profile in the *DNSEventTraceTransformed* data set. *ISARCRResultSetASimulation* produces a connected state simulation that gears closers towards 58% successful connections and 42% unsuccessful for a sample size of 10000 mea-

surement events. In the earlier section on significant FTA DNS parameters, section 5.6.3, this statistic is present in Pang’s primary data set.

- The `np.mean(ISARCResultSetASimulation[2])` at first simulation presented a mean of 42.1548, and
- an `np.mean(ISARCResultSetASimulation[4])` at first simulation produced a mean of 57.8452.

The updated dataset executed on 25 April 2021 produced 15 simulations containing five 10000 sample1 entries, and five 20000 sample1 entries, and 5 50000 sample1 entries. A full sample2 population (384991) was used. The results yield a similar error margin between sample1 (DNSConnectionEventTraceGenerated) and sample2 (DNSEventTraceTransformed) date sets.

- The `np.mean(ISARCResultSetASimulation[2])` presents a mean of 41.9131, and
- an `np.mean(ISARCResultSetASimulation[4])` produces a mean of 58.0869.

Table 5.12: Original Simulation Results for Set A: Population 10000

FCC	FCP	SCC	SCP	Match	ISARC Sample	FTA Sample
733	43.6050	948	56.3950	1681	10000	10000
596	41.6492	835	58.3508	1431	10000	10000
695	42.7429	931	57.2571	1626	10000	10000
717	43.7195	923	56.2805	1640	10000	10000
677	42.9024	901	57.0976	1578	10000	10000
622	42.1695	853	57.8305	1475	10000	10000
637	38.4894	1018	61.5106	1655	10000	10000
605	40.7407	880	59.2593	1485	10000	10000
678	43.5453	879	56.4547	1557	10000	10000
673	41.9838	930	58.0162	1603	10000	10000

Table 5.12 presents the first simulation set containing 10000 samples for both sample1 and sample2. The output of `_ResultSetANative` code block full automates the simulation output. The original sampled output appears inline in the text, however, the continuous simulation of `_ResultSetANative` yields a consistent Failed Connection Count (FCC) and Success Connection Count (SCC). As noted in the first simulation, the average of the Failed Connection Percentage (FCP) and Success Connection Percentage (SCP) yielded 42.1548 and 57.8452, respectively. Additionally, an additional run of sample1 and sample2 comparison yields an FCP of 41.9131 and an SCP of 58.0869. A rounded output value yields a value close to the the 42% `{0:Unavailability}` and 58% `{1:Availability}` statistic found in the FTA. Continuous update to the ISARC results summary locates in the `ISARCResultSetASimulation.csv` file in the `isarcSimulations` folder.

Table 5.13: ISARC Result Set Field Descriptions

Column	Description
FCC	This field contains the Failed Connection Count (FCC) for the <i>Front-end System Set</i> towards an LDNS node.
FCP	This field contains the percentage of FCC in accordance to the matched records between the ISARC generated set and the FTA data set.
SCC	This field contains the Success Connection Count (SCC) for the <i>Front-end System Set</i> towards an LDNS node.
SCP	This field contains the percentage of SCC in accordance to the matched records between the ISARC generated set and the FTA data set.
Match	This field contains the total number of matching nodes between the sampled ISARC generated set (<i>DNSConnectionEventTraceGenerated</i>) and the sampled FTA data set (<i>DNSEventTraceTransformed</i>).
ISARC Sample	This field contains the number of sampled records from <i>DNSConnectionEventTraceGenerated</i> . The upper limit on the number of records is unlimited given the analysis requirement, storage, processing and other data handling constraints.
FTA Sample	This field contains the number of sampled records from <i>DNSEventTraceTransformed</i> . The upper limit on the number of records is a constant 384991 as per the LDNS data set in Pang et al. (2004), extracted from Epema et al. (2014).

The fields descriptions for Table 5.12 is shown in Table 5.13.

The output of Simulation Sample Set 2 results from the execution of 10 trials for *resultsSetASampleSize* where *DNSConnectionEventTraceGenerated* = 20000 and *DNSEventTraceTransformed* = 20000. The inline results produces an extension to the original simulation set. For continuous testing the inline code blocks produce results that fall within the appropriate errors claimed in the results summary section. Table 5.14 contains the extended original simulation set results for simulation sample set 2.

- an `np.mean(ISARCResultSetASimulation[2])` of 42.2334, and
- an `np.mean(ISARCResultSetASimulation[4])` of 57.7666.

The output of Simulation Sample Set 3 results from the execution of 10 trials for a fixed sample set of 50000 samples for both *Front-end System Set* and LDNS that produces an;

- `np.mean(ISARCResultSetASimulation[2])` of 42.0687, and
- an `np.mean(ISARCResultSetASimulation[4])` of 57.9313.

Tables that presents a consistent *Front-end System Set* to LDNS connection pool sample size is shown in Tables 5.12, 5.14, and 5.15.

The output of Simulation Sample Set 4 results from the execution of 20 trials. It contains a fixed sample set of 50000 samples for *Front-end System Set* and the maximum sample size of 384991 for the LDNS that produces an;

Table 5.14: Simulation Results for Set A: Population 10000 & 20000

FCC	FCP	SCC	SCP	Match	ISARC Sample	FTA Sample
733	43.6050	948	56.3950	1681	10000	10000
596	41.6492	835	58.3508	1431	10000	10000
695	42.7429	931	57.2571	1626	10000	10000
717	43.7195	923	56.2805	1640	10000	10000
677	42.9024	901	57.0976	1578	10000	10000
622	42.1695	853	57.8305	1475	10000	10000
637	38.4894	1018	61.5106	1655	10000	10000
605	40.7407	880	59.2593	1485	10000	10000
678	43.5453	879	56.4547	1557	10000	10000
673	41.9838	930	58.0162	1603	10000	10000
2687	42.4018	3650	57.5982	6337	20000	20000
2628	42.2916	3586	57.7084	6214	20000	20000
2629	41.9566	3637	58.0434	6266	20000	20000
2576	41.6694	3606	58.3306	6182	20000	20000
2608	41.3312	3702	58.6688	6310	20000	20000
2705	43.6290	3495	56.3710	6200	20000	20000
2761	43.6039	3571	56.3961	6332	20000	20000
2579	40.8199	3739	59.1801	6318	20000	20000
2697	42.9527	3582	57.0473	6279	20000	20000
2629	42.4649	3562	57.5351	6191	20000	20000

Table 5.15: Simulation Results for Set A: Population 10000, 20000 & 50000

FCC	FCP	SCC	SCP	Match	ISARC Sample	FTA Sample
733	43.6050	948	56.3950	1681	10000	10000
596	41.6492	835	58.3508	1431	10000	10000
695	42.7429	931	57.2571	1626	10000	10000
717	43.7195	923	56.2805	1640	10000	10000
677	42.9024	901	57.0976	1578	10000	10000
622	42.1695	853	57.8305	1475	10000	10000
637	38.4894	1018	61.5106	1655	10000	10000
605	40.7407	880	59.2593	1485	10000	10000
678	43.5453	879	56.4547	1557	10000	10000
673	41.9838	930	58.0162	1603	10000	10000
2687	42.4018	3650	57.5982	6337	20000	20000
2628	42.2916	3586	57.7084	6214	20000	20000
2629	41.9566	3637	58.0434	6266	20000	20000
2576	41.6694	3606	58.3306	6182	20000	20000
2608	41.3312	3702	58.6688	6310	20000	20000
2705	43.6290	3495	56.3710	6200	20000	20000
2761	43.6039	3571	56.3961	6332	20000	20000
2579	40.8199	3739	59.1801	6318	20000	20000
2697	42.9527	3582	57.0473	6279	20000	20000
2629	42.4649	3562	57.5351	6191	20000	20000
16289	41.7827	22696	58.2173	38985	50000	50000
16284	41.5419	22915	58.4581	39199	50000	50000
16251	41.9543	22484	58.0457	38735	50000	50000
16523	41.6595	23139	58.3405	39662	50000	50000
16150	42.0234	22281	57.9766	38431	50000	50000
16178	41.5940	22717	58.4060	38895	50000	50000
16481	41.9236	22831	58.0764	39312	50000	50000
16430	42.0646	22629	57.9354	39059	50000	50000
16099	41.6382	22565	58.3618	38664	50000	50000
16124	41.2115	23001	58.7885	39125	50000	50000

- $\text{np.mean}(\text{ISARCResultSetASimulation}[2])$ of 42.0216, and
- an $\text{np.mean}(\text{ISARCResultSetASimulation}[4])$ of 57.9784.

Table 5.16: Simulation Results for Set A: Population ED 50000 FTA 384991

FCC	FCP	SCC	SCP	Match	ISARC Sample	FTA Sample
733	43.6050	948	56.3950	1681	10000	10000
596	41.6492	835	58.3508	1431	10000	10000
695	42.7429	931	57.2571	1626	10000	10000
717	43.7195	923	56.2805	1640	10000	10000
677	42.9024	901	57.0976	1578	10000	10000
622	42.1695	853	57.8305	1475	10000	10000
637	38.4894	1018	61.5106	1655	10000	10000
605	40.7407	880	59.2593	1485	10000	10000
678	43.5453	879	56.4547	1557	10000	10000
673	41.9838	930	58.0162	1603	10000	10000
2687	42.4018	3650	57.5982	6337	20000	20000
2628	42.2916	3586	57.7084	6214	20000	20000
2629	41.9566	3637	58.0434	6266	20000	20000
2576	41.6694	3606	58.3306	6182	20000	20000
2608	41.3312	3702	58.6688	6310	20000	20000
2705	43.6290	3495	56.3710	6200	20000	20000
2761	43.6039	3571	56.3961	6332	20000	20000
2579	40.8199	3739	59.1801	6318	20000	20000
2697	42.9527	3582	57.0473	6279	20000	20000
2629	42.4649	3562	57.5351	6191	20000	20000
16289	41.7827	22696	58.2173	38985	50000	50000
16284	41.5419	22915	58.4581	39199	50000	50000
16251	41.9543	22484	58.0457	38735	50000	50000
16523	41.6595	23139	58.3405	39662	50000	50000
16150	42.0234	22281	57.9766	38431	50000	50000
16178	41.5940	22717	58.4060	38895	50000	50000
16481	41.9236	22831	58.0764	39312	50000	50000
16430	42.0646	22629	57.9354	39059	50000	50000
16099	41.6382	22565	58.3618	38664	50000	50000
16124	41.2115	23001	58.7885	39125	50000	50000
126551	41.9326	175245	58.0674	301796	50000	384991
126909	41.9559	175573	58.0441	302482	50000	384991
125792	41.8991	174434	58.1009	300226	50000	384991
128071	42.0183	176727	57.9817	304798	50000	384991
124962	41.8501	173632	58.1499	298594	50000	384991
126814	41.9419	175542	58.0581	302356	50000	384991
126554	41.9435	175171	58.0565	301725	50000	384991
126329	41.9217	175016	58.0783	301345	50000	384991
125398	41.8804	174021	58.1196	299419	50000	384991
126307	41.9197	175000	58.0803	301307	50000	384991
125815	41.8934	174507	58.1066	300322	50000	384991
128898	42.0617	177552	57.9383	306450	50000	384991
126698	41.9496	175326	58.0504	302024	50000	384991
127421	41.9792	176113	58.0208	303534	50000	384991
129122	42.0689	177808	57.9311	306930	50000	384991
126201	41.9169	174873	58.0831	301074	50000	384991
126681	41.9380	175386	58.0620	302067	50000	384991
128833	42.0572	177495	57.9428	306328	50000	384991
126569	41.9362	175244	58.0638	301813	50000	384991
126792	41.9508	175448	58.0492	302240	50000	384991

To recall, the population statistics for the main data set *DNSEventTraceTransformed* shows; 223596 individually occurring `{1:Availability}` states and 161395 individually occurring `{0:Unavailability}` event states. Thus, a selection for the population set of 384991 events can be split into 58% `{1:Availability}` and 42% `{0:Unavailability}`. The population split in the code block assumes - with all things equal - that a connected state will prevail 58% of the time

within each validation sample set. In summary, 50 trial simulations that executes with an increasing static sample sizes for **Front-end System Set** and FTA yields the results in Table 5.17.

Result Set A, which contains Simulation Samples 1 to 4, indicates that a simplified non-attached relational model provides some reprieve when testing against the historical data set. This test takes into account only the **Addressing** parameter by building into a match between generated address and the address encountered in the historical data set. Figure 5.17 summarises the impact of the first ISARC test set.

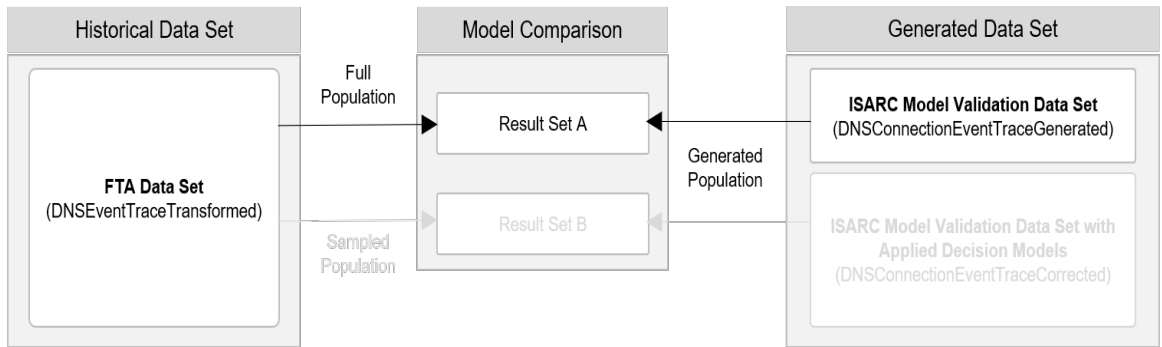


Figure 5.17: Results Summary A Overview

Table 5.17: Simulation Results for Set A: Target Overview

SampleSizes	FCP	SCP	FCPTarget	SCPTarget	FCPMargin	SCPMargin
10000/10000	42.1548	57.8452	41.9218	58.0782	-0.233017	0.233017
20000/20000	42.2334	57.7666	41.9218	58.0782	-0.311676	0.311676
50000/50000	42.0687	57.9313	41.9218	58.0782	-0.146988	0.146988
50000/384991	42.0216	57.9784	41.9218	58.0782	-0.0997947	0.0997947

A code block that realises Table 5.17 uses the actualised values from averages attained from Tables 5.12, 5.14, 5.15, and 5.16. The FCPMargin and SCPMargin output field contain values that do not show significant precision values, hence the significant values are indicated in the code block below.

Listing 5.25: Original Simulation Results for Set A: Target Table Realisation

```

1 #Original Simulation Result Summary Realisation
2
3 ISARCResultSetASimTable = Table().with_columns(
4     "FTA_EDGWSample", make_array("10000/10000", "20000/20000", "50000/50000", "384991/50000"),
5     "IncrementalFCP", make_array(42.15477591708826, 42.2334348336425, 42.06874706712469, 42.02155395232509),
6     "IncrementalSCP", make_array(57.84522408291173, 57.76656516635751, 57.9312529328753, 57.97844604767491),
7     "FTAFCPTarget", make_array(41.92175920995555, 41.92175920995555, 41.92175920995555, 41.92175920995555),
8     "FTASCPTarget", make_array(58.07824079004445, 58.07824079004445, 58.07824079004445, 58.07824079004445),
9 )
10 ISARCResultSetASimTable = ISARCResultSetASimTable.with_columns(
11     "FCPMargin", ISARCResultSetASimTable['FTAFCPTarget'] - ISARCResultSetASimTable['IncrementalFCP'],
12     "SCPMargin", ISARCResultSetASimTable['FTASCPTarget'] - ISARCResultSetASimTable['IncrementalSCP'],
13 )
14 ISARCResultSetASimTable

```

Table 5.17 is further extensible to include variations of sample sizes. Once confirmed that the output of the *_GlobalDataSetGenerator* - the *DNSConnectionEventTraceGenerated* data set - represents outputs that fall within acceptable margins of the FTA targets, the ISARC Decision Model output aims to optimize within the preferred increase and decrease with the primary target range. As indicated in Table 5.17, the target margins for the current sample set ranges from -0.311676 to 0.311676.

The Failed Connection Percentage (FCP) and Success Connection Percentage (SCP) are significant in the context of the data generation requirements for the ISARC. The FCP and SCP present a range similar to the fixed availability and unavailability statistics in the FTA data set. It is significant in that increasing sample sizes produce a suitably margined FCP and SCP without declaring the fixed split of 42

5.10.2 Result Set B Summary

The results from the previous null hypothesis in *Result Set A* - nothing besides random behaviour - delivers a data set that mimics the FTA event measurements for **{0:Unavailability}** and **{1:Availability}** connection state averages within a reasonable margin. This output reflects a typical sampling output related to the 42%/58% ratio found in the original data set by Pang et al. (2004). This approach indicates that several constraints reduce the original population sample focus on randomisation of the subset of DNS servers probed in the Pang study. Pang et al. (2004) references the “PASTA” principle, where the ability to observe each host’s availability and unavailability would approximate the amount of time

the host spends in each respective state. Increases in fixed-rate measurements by Pang indicate an identical availability distribution by increasing the number of measurement probes. Compared to experiments with larger intervals, the increased fixed rate yielded a "nearly identical" availability profile in Pang et al. (2004).

Hence, the 42%/58% split represented in the master DNSEventTraceTransformed LDNS data set serves as a primary reliability statistic. Code block *_ResultSetBCorrected* creates the Result Set B data set using the samples and output used in Result Set A.

Listing 5.26: Result Set B: Fixed ISARC Decision Model Output

```

1  #IoT Systems Architecture for Runtime Configuration Results Sample
2  #Code Block: \_ResultSetBCorrected
3  #Output: ISARCResultSetBDetail
4  #Output: ISARCResultSetBSummary
5  #Output: ISARCResultSetBSimulation
6
7  #Simulation Approach;
8  #1. Use Sample 1 and Sample 2 from Results Sample
9
10 import datetime as dt
11
12 #Create replicas of ISARC sample data sets
13 sampleB1, sampleB2, ISARCResultSetB = sample1, sample2, ISARCResultSetA
14 isarcConfigTableB = isarcConfigTable
15
16 #Fixed Time Step Decision. Rerun with new timeStep values 660, 720, 780, 6000, 6660 & 5400
17 resultSetBOutputSample = 50000
18 timeStep = 600
19
20 #Create Unique Data Set for ISARCResultSetB
21 ISARCResultSetB_ = ISARCResultSetB
22
23 #Apply Datetime format transformations on ISARCResultSetB_
24 import pandas as pd
25 a=pd.to_datetime(ISARCResultSetB_['DNSStateStart'])
26 dt1 = np.array(a,dtype=np.datetime64)
27 ts1 = pd.DatetimeIndex([dt1])[0]
28
29 b=pd.to_datetime(ISARCResultSetB_['DNSStateStop'])
30 dt2 = np.array(b,dtype=np.datetime64)
31 ts2 = pd.DatetimeIndex([dt2])[0]
32
33 ts3 = pd.DatetimeIndex(np.array(pd.to_datetime(ISARCResultSetB_['EDGWConnEventStart']),dtype=np.datetime64))
34 ts4 = pd.DatetimeIndex(np.array(pd.to_datetime(ISARCResultSetB_['EDGWConnEventStop']),dtype=np.datetime64))
35
36 #Insert Datetime formatted columns into ISARCResultSetB_
37 ISARCResultSetB_ = ISARCResultSetB_.with_column(
38     "DNSStateStart_", ts1
39 )
40 ISARCResultSetB_ = ISARCResultSetB_.with_column(
41     "DNSStateStop_", ts2
42 )
43 ISARCResultSetB_ = ISARCResultSetB_.with_column(
44     "EDGWConnEventStart_", ts3
45 )
46 ISARCResultSetB_ = ISARCResultSetB_.with_column(
47     "EDGWConnEventStop_", ts4
48 )

```

```

49 |
50 | #Create Formatted Data Set with correct Datetime Formatted Columns from ISARCTestSetB_ into
    | ISARCTestSetBDetail
51 | ISARCTestSetBDetail = ISARCTestSetB_.select(
52 |     "RowID",
53 |     "NodeName",
54 |     "CandidateDNS",
55 |     "EDGWConnEventStart_",
56 |     "EDGWConnEventStop_",
57 |     "DNSEventID",
58 |     "DNSEventState",
59 |     "DNSStateStart_",
60 |     "DNSStateStop_",
61 |     "DNSEventStatePeriod",
62 |     "ConnectionPossibility"
63 | )
64 |
65 | eventSec = np.vectorize(lambda x: x.total_seconds())
66 |
67 | #Create Timedelta Columns for conditional logic in ISARCTestSetBDetail
68 | ISARCTestSetBDetail = ISARCTestSetBDetail.with_column(
69 |     "EventStartTimeDelta", eventSec(ISARCTestSetBDetail[7] - ISARCTestSetBDetail[3])
70 | )
71 | ISARCTestSetBDetail = ISARCTestSetBDetail.with_column(
72 |     "EventStopTimeDelta", eventSec(ISARCTestSetBDetail[8] - ISARCTestSetBDetail[4])
73 | )
74 |
75 | #Create updated decision function to apply in ISARCTestSetBDetail
76 | def setUpdatedDecisionModel(modelSensitivity, dnsState):
77 |     if modelSensitivity > timeStep * -1 and modelSensitivity < 0 and dnsState == 1:
78 |         return 'Success'
79 |     elif modelSensitivity >= 0:
80 |         return 'Out of Range'
81 |     else: return 'Out of Range'
82 |
83 | #Create final data set ISARCTestSetB from ISARCTestSetBDetail
84 | ISARCTestSetB = ISARCTestSetBDetail.with_column('ConnectionPossibilityAlt', ISARCTestSetBDetail.apply(
    |     setUpdatedDecisionModel, 'EventStartTimeDelta', 'DNSEventState'))
85 | ISARCTestSetBSummary = ISARCTestSetB.group("ConnectionPossibilityAlt")
86 | ISARCTestSetBSummary = ISARCTestSetBSummary.with_column('SimulationTime', pd.to_datetime('now'))
87 | ISARCTestSetBSummary = ISARCTestSetBSummary.with_column('SimulationTimestep', timeStep)
88 | ISARCTestSetBSummary = ISARCTestSetBSummary.with_column('ConnectionMatch', ISARCTestSetB.num_rows)
89 | ISARCTestSetBSummary = ISARCTestSetBSummary.with_column(
90 |     "PercentageSplit", ISARCTestSetBSummary.apply(lambda x, y: (x / y) * 100, 'count', 'ConnectionMatch')
91 | )
92 |
93 | #Produce Scatter from ISARCTestSetB
94 | ISARCTestSetBScatterDetail = ISARCTestSetB.where('ConnectionPossibilityAlt', are.not_equal_to('Out of Range'))
95 | ISARCTestSetBSummaryS = Table().with_columns([
96 |     'StartDelta', ISARCTestSetBScatterDetail['EventStartTimeDelta'],
97 |     'DNSEventPeriod', ISARCTestSetBScatterDetail['DNSEventStatePeriod'],
98 | ])
99 | ISARCTestSetBSummaryS.scatter('StartDelta', fit_line = True)
100 |
101 | ISARCTestSetBSimulation = ISARCTestSetBSimulation.append(ISARCTestSetBSummary)
102 | ISARCTestSetBSimulation.to_csv('.\isarcSimulations\ISARCTestSetBSimulation.csv')
103 | ISARCTestSetBSummary.to_csv('.\isarcSimulations\ISARCTestSetBSummary.csv')

```

The *_ResultSetBCorrected* code block delivers Result Set B that ensures that a stable input exhibits the output of the decision variations on the primary sampled data set. Care is taken to ensure that the Result Set A output remains consistent as if the matching of the

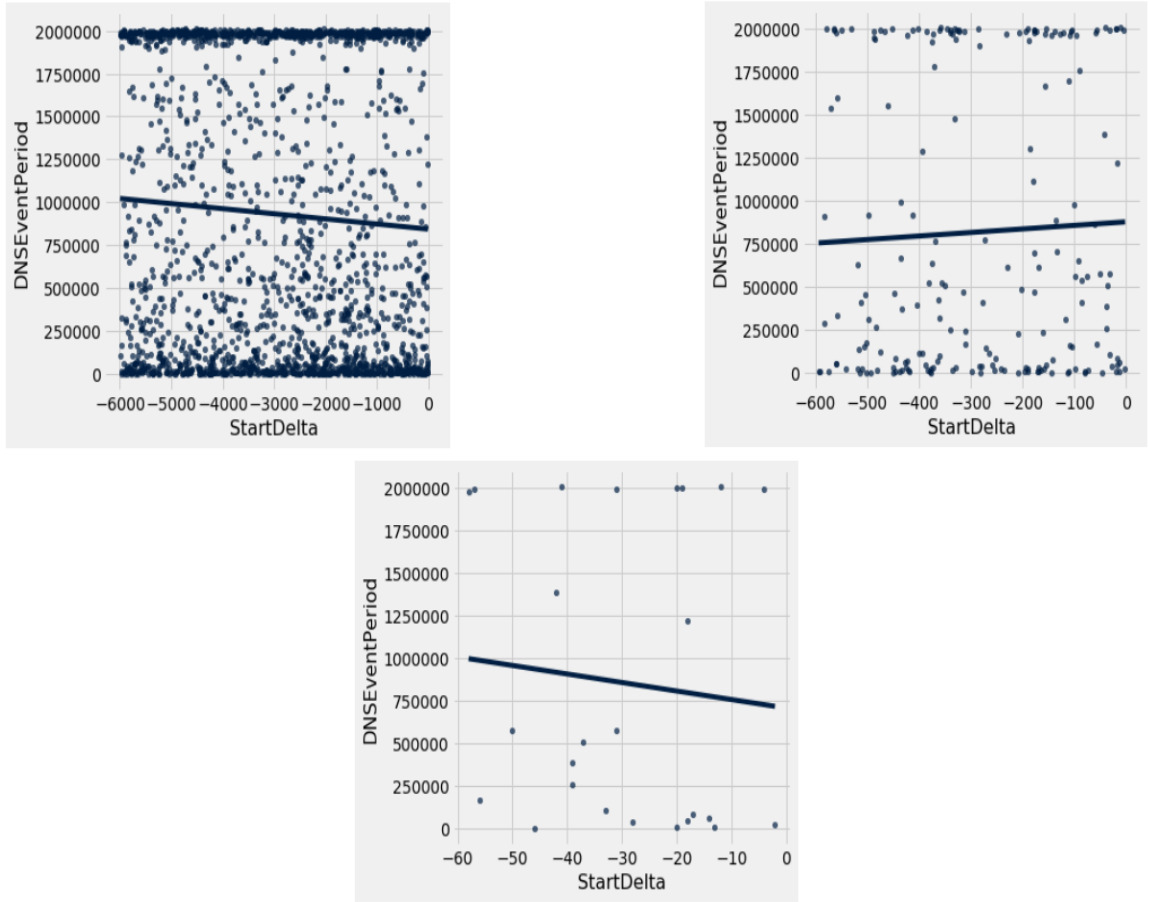


Figure 5.18: Results Set B Sample Summary Overview

candidate DNS node and actual DNS node occurred during the execution of the result set A output. Result Set A already indicates that a single comparison field delivers matching of random `Front-end System Set` node within a suitable FTA measurement margin.

Figure 5.18 presents the output of the `_ResultSetBCorrected` code block as three consistent yet separate executions for a more accurate view of the matched `Front-end System Set` and FTA DNS components. The `DNSEventPeriod` axis displays the difference between Event Stop Time and Event Start Time in the FTA data set - in seconds. The `StartDelta` axis displays the difference between the start of the generated `Front-end System Set` events and the actual FTA DNS event - in seconds. Result Set A produced a match of the IP address between the generated LDNS node and the actual LDNS node, creating an output that mimics the 42%/58% statistic in the main FTA data set. Result Set B uses the sample output from Result Set A and contains a conditional time delta verification for each event within 6000-second, 600-second, and 60-second boundaries. The additional timestep check reduces the number of matches expected in a pure random test due to a larger population

of coupled parameters such as event time in seconds and the LDNS node address. A lower time step indicates a more sparse connection possibility, while a longer timestep such as the 100-minute limit yields a higher density population. This connection density is visible in the top left scatter of Figure 5.18 versus the lower scatter chart, which indicates fewer connection successes. It stands to reason that a lengthier time step will aid in identifying a greater number of matches. A lower time step will attempt to match a narrower condition as the generated data set's node addresses and event time only link to the historical data set. The historical data set only applies to constraining the node name and event time ranges.

Each event in the sample sets generated by the three boundaries has a reduced chance of matching the candidate LDNS node to an exact period of the event exhibited in the FTA events. The `_ResultSetBCorrected` code block presents a greater depth of the ISARC data generation process showing the truly random nature of the ISARC decision model outputs. It considers one additional parameter - the time delta - that produces increased accuracy for matching the generated *DNSTConnectionEventTraceGenerated* data set and the master *DNSEventTraceTransformed*. The number of matches decreases to a percentage outside of the allowable FTA margins. The target optimisation for the `_ResultSetBCorrected` code block is to apply the decision model configuration outputs such that it provides a stepped increase towards a suitable FTA measurement margin.

Figure 5.15 in section 5.9.2 outputs from the earlier code block `_ISARCDecisionParameterSelection`, presents the outcome of a 100000 cycle simulation set containing a random sample of ten individual parameters from the ISARC parameter configuration table. The result indicates that the consideration for lower decision sensitivity on parameters yields a more significant number of non-action considerations. This approach yields a no better result than a pure non-deterministic view of architecture configuration variation handling. Decreasing most parameter values to equate to 0.001 produces no better selection criteria and no increase in decision effectiveness. Result Set B scales different architectural options to tune FCP and SCP values within the FTAFCP and FTASCP margins by setting higher sensitivity for individual parameters.

The outcome of single and multiple parameter selection will establish an increase or decrease of connection states within the previously sampled outputs. The intent that a specific set of configurations yield a higher determinism in the decision process indicates that a purposeful intent is present to ensure that the decision space is stipulated. Furthermore, configuration decisions guide changes in significant quantitative measures. Finally, decisions about

function and form appear outside the realm of design decisions. These tenets appear in structured systems architecture, Crawley et al. (2016)(pp.326-327). The converged ISARC decision configuration table (*isarcConfigTable*) contains 77 individual parameters. By applying a simplified view of varying runtime architecture configurations for increased connection possibility, all parameters in the ISARC Decision Model 1 to 4 could manifest as equally important or unimportant. The former does not apply the appropriate boundary considerations, and the latter does not create a position any better than a pure non-deterministic view of the IoT-Ce.

An applied PI control strategy that represents an instance of the [IoTSystemControlHandling](#) function aims to reduce the {0:Unavailability} states over the measurement period. The lower and upper range from **Result Set A** is **(-0.311676 to 0.311676)**. Result set A provides a value closer in proximity to the FTA availability and unavailability metric. For the ISARC decision model, the intersection of each significant parameter in the decision DSM is an immutable value of 1, and all other intersection values tend towards or away from 1. The handling of individual or clustered sensitivities delivers the core phasing of a decision model that holds flexible values in a limited yet credible range. The row and column values in the decision model DSM is extensible to include pathways to sensitivities or cumulative handling of decision aggregate values. DSM pathways manifest when using DSM to compute the reachability of parameters, as indicated in Eppinger and Browning (2012)(pp.134). A series of custom functions that set and determine the SValues appear in the code block below. These functions produce the executable control functions that modify the results in subsequent results detail and summaries.

Listing 5.27: ISARC Decision Model Control Functions

```

1 #IoT Systems Architecture for Runtime Configuration Results Sample
2 #Code Block: \_ResultSetBCorrected_AppliedControl
3 #Output: ISARCResultSetBSimulationComparison_AppliedControl
4 #Master Data Set Import for Primary Sampling Proof of Control Decision Effectiveness
5
6 import pandas as pd
7 import numpy as np
8 from datetime import datetime, timezone, time
9 from datascience import *
10
11 now = datetime.now()
12
13 import matplotlib
14 matplotlib.use('Agg')
15 from datascience import Table
16 %matplotlib inline
17 import matplotlib.pyplot as plt
18 plt.style.use('fivethirtyeight')
19 import plotly.graph_objects as go
20 from simple_pid import PID
21
22 #Pang DNS Failure Trace Data Set
23 traceds = pd.read_csv('event_trace.csv')

```

```

24 |
25 | #####
26 | #ISARC Control Parameters#####
27 | #####
28 | SampleSize = 5000
29 | KProp      = 1
30 | KIntegr    = 1.5
31 | KDeriv     = 0
32 | SetPoint   = 1
33 | #####
34 | #####
35 |
36 | #Analysis Table Data Set for Full Event Trace Population
37 | eventds = Table().with_columns('EventID',traceds.event_id)
38 | eventds = eventds.with_columns('NodeName',traceds.node_name)
39 | eventds = eventds.with_columns('EventStart',traceds.event_start_time)
40 | eventds = eventds.with_columns('EventStartE',traceds.event_start_time)
41 | eventds = eventds.with_columns('EventStop',traceds.event_stop_time)
42 | eventds = eventds.with_columns('EventStopE',traceds.event_stop_time)
43 | eventds = eventds.with_columns('EventType',traceds.event_type)
44 | eventds = eventds.with_columns('EventTimeDiff', traceds['event_stop_time'] - traceds['event_start_time'])
45 |
46 | eventds['EventStart'] = pd.to_datetime(eventds['EventStart'],unit='s')
47 | eventds['EventStop'] = pd.to_datetime(eventds['EventStop'],unit='s')
48 |
49 | ISARCResultSetBSimulationComparison_AppliedControl = Table(make_array("SimulationTime","EventStart","
    FTASampleFC","FTASampleSC","FTASimulationSample","FTASampleFCP","FTASampleSCP","FTA Unavailability Actual"
    ,"FTA Availability Actual","ISARC Unavailability Controlled","ISARC Availability Controlled"))
50 |
51 | #Get Nodes that have 501 to 600 event activity
52 | MostActiveNodeDetail = pd.read_csv('DNSMostActiveNodeDetail.csv')
53 | MostActiveNodeDetail_ = Table().with_columns([
54 |     'NodeName', MostActiveNodeDetail['NodeName'],
55 |     'Unavailable', MostActiveNodeDetail['0'],
56 |     'Available', MostActiveNodeDetail['1'],
57 |     'Event Summary', MostActiveNodeDetail['EventSum'],
58 |     'Event Summary Category', MostActiveNodeDetail['EventSumCategory'],
59 | ])
60 | SelectedNodes = MostActiveNodeDetail_.where('Event Summary Category', are.equal_to("501-600"))["NodeName"]
61 |
62 | # Applied IoTSystemControlHandling Function to Mimic Control Decision in ISARC
63 | def IoTSystemControlHandling(nodeName, allowedNodeList):
64 |     if nodeName in allowedNodeList:
65 |         controlReason = "Control suggested in this Node"
66 |     else:
67 |         controlReason = "Control not suggested in this Node"
68 |     return controlReason
69 |
70 | # Setting Gain & Setpoint
71 | pid = PID(Kp=KProp, Ki=KIntegr, Kd=KDeriv, setpoint=SetPoint)
72 | sample_size = SampleSize
73 |
74 | controlled_system = Table().with_columns(
75 |     'NodeName', eventds[1],
76 |     'EventStart',eventds[2],
77 |     'NodeState', eventds[6]
78 | )
79 |
80 | controlled_sample = controlled_system.sample(sample_size, with_replacement=False)
81 | eventSDate = []
82 | for _ in np.arange(controlled_sample.num_rows):
83 |     eventSDate = np.append(eventSDate, controlled_sample[1][_].date().strftime('%Y-%m-%d'))
84 |
85 | controller_input = Table().with_columns([
86 |     'NodeName',    controlled_sample[0],

```

```

87     'EventStart', eventSDate,
88     'NodeState', controlled_sample[2]
89 ])
90
91 controller_output = controller_input.copy()
92 controller_input_1 = controller_input.copy()
93
94 for _ in np.arange(controller_output.num_rows):
95     if controller_output[2][_] == 0 and controller_output['NodeName'][_] in SelectedNodes:
96         control = pid(controller_output[2][_])
97         controller_output[2][_] = control
98
99 controlled_system = Table().with_columns([
100     "NodeName", controller_input['NodeName'],
101     "EventStart", controller_input['EventStart'],
102     "FTANodeMasterState", controller_input['NodeState'],
103     "ISARCControlledState", controller_output['NodeState']
104 ])
105
106 ctlReas = []
107 for _ in np.arange(controlled_system.num_rows):
108     ctlCurrReas = IoTSystemControlHandling(controlled_system["NodeName"][_], SelectedNodes)
109     ctlReas = np.append(ctlReas, ctlCurrReas)
110
111 controlled_system = controlled_system.with_columns([
112     "ISARCControlReason", ctlReas
113 ])
114
115 scatter_base_r = controlled_system.pivot('FTANodeMasterState', 'EventStart')
116 scatter_base_c = controlled_system.pivot('ISARCControlledState', 'EventStart')
117
118 x = scatter_base_r['EventStart']
119 yr = scatter_base_r['0']
120 zr = scatter_base_r['1']
121 yc = scatter_base_c['0']
122 zc = scatter_base_c['1']
123
124 fig = go.Figure()
125 fig.add_trace(go.Scatter(x=x, y=yr,
126                         marker_color=yr,
127                         mode='lines+markers',
128                         name='FTA Unavailability (Actual)'))
129 fig.add_trace(go.Scatter(x=x, y=zr,
130                         marker_color=zr,
131                         mode='lines+markers',
132                         name='FTA Availability (Actual)'))
133 fig.add_trace(go.Scatter(x=x, y=yc,
134                         marker_color=yc,
135                         mode='lines+markers',
136                         name='ISARC Unavailability (Controlled)'))
137 fig.add_trace(go.Scatter(x=x, y=zc,
138                         marker_color=zc,
139                         mode='lines+markers',
140                         name='ISARC Availability (Controlled)'))
141 fig.update_xaxes(
142     tickangle = 0,
143     nticks = len(x),
144     title_text = "Event Date",
145     title_font = {"size": 15},
146     title_standoff = 25)
147 fig.update_yaxes(
148     tickangle = 0,
149     title_text = "Event Count Summary",
150     title_font = {"size": 15},
151     title_standoff = 25)

```

```

152 fig.write_html('ISARCControllerModification.html', auto_open=False)
153
154 ISARCTestResultSetBSimulation_AppliedControl = Table().with_columns([
155     "EventStart", scatter_base_r['EventStart'],
156     "FTA Unavailability Actual", scatter_base_r['0'],
157     "FTA Availability Actual", scatter_base_r['1'],
158     "ISARC Unavailability Controlled", scatter_base_c['0'],
159     "ISARC Availability Controlled", scatter_base_c['1'],
160 ])
161 ISARCTestResultSetBSimulation_AppliedControl = ISARCTestResultSetBSimulation_AppliedControl.with_columns([
162     "SimulationTime", pd.to_datetime('now'),
163     'FTASampleFC', np.int(controlled_system.group('FTANodeMasterState').where('FTANodeMasterState', are.
164     equal_to(0))['count']),
164     'FTASampleSC', np.int(controlled_system.group('FTANodeMasterState').where('FTANodeMasterState', are.
165     equal_to(1))['count']),
165     "FTASimulationSample", np.sum(controlled_system.group('FTANodeMasterState')['count']),
166 ])
167 ISARCTestResultSetBSimulation_AppliedControl = ISARCTestResultSetBSimulation_AppliedControl.with_columns([
168     'FTASampleFCP', ISARCTestResultSetBSimulation_AppliedControl.apply(lambda x, y: (x / y) * 100, 'FTASampleFC', "
169     FTASimulationSample"),
169     'FTASampleSCP', ISARCTestResultSetBSimulation_AppliedControl.apply(lambda x, y: (x / y) * 100, 'FTASampleSC', "
170     FTASimulationSample"),
171 ])
172 ISARCTestResultSetBSimulation_AppliedControl = ISARCTestResultSetBSimulation_AppliedControl.select([
173     "SimulationTime",
174     "EventStart",
175     "FTASampleFC",
176     "FTASampleSC",
177     "FTASimulationSample",
178     "FTASampleFCP",
179     "FTASampleSCP",
180     "FTA Unavailability Actual",
181     "FTA Availability Actual",
182     "ISARC Unavailability Controlled",
183     "ISARC Availability Controlled"
184 ])
185 ISARCTestResultSetBSimulationComparison_AppliedControl = ISARCTestResultSetBSimulationComparison_AppliedControl.append(
186     ISARCTestResultSetBSimulation_AppliedControl)
187 ISARCTestResultSetBSimulationComparison_AppliedControl.to_csv(".\isarcSimulations\ISARCTestResultBSimulation_Corrected_
    " + now.strftime("%m%d%Y_%H%M%S") + ".csv")

```

A standard Python library that handles proportional and integral control is used to outline the impact of suggested suggesting control on a node that is in `{0:Unavailability}` state, Lundberg (2018). The applied PI controller with a configuration of $PID(K_p=>0, K_i>=0, K_d=0.0, setpoint=1)$ yields a consistent output that applies control within the cluster of high interaction DNS nodes. These nodes contain 501 to 600 interactions. The controller effectively reduces the `{0:Unavailability}` state aggregate count over the measurement period, however given the variation uniform random sampling approach, 5000 continuous samples are tested. The output shape of the unavailability metric in the FTA data set (TraceDS, EventDS, and labelled earlier as *DNSEventTraceTransformed*) yields a consistent 42%/58% FCP and SCP metric. The output in Figure 5.19 indicates a marginal reduction of the unavailability states if a decision is to apply a control that changes the event's state from `{0:Unavailability}` to `{1:Availability}`. This is inline with a decision to protect connection

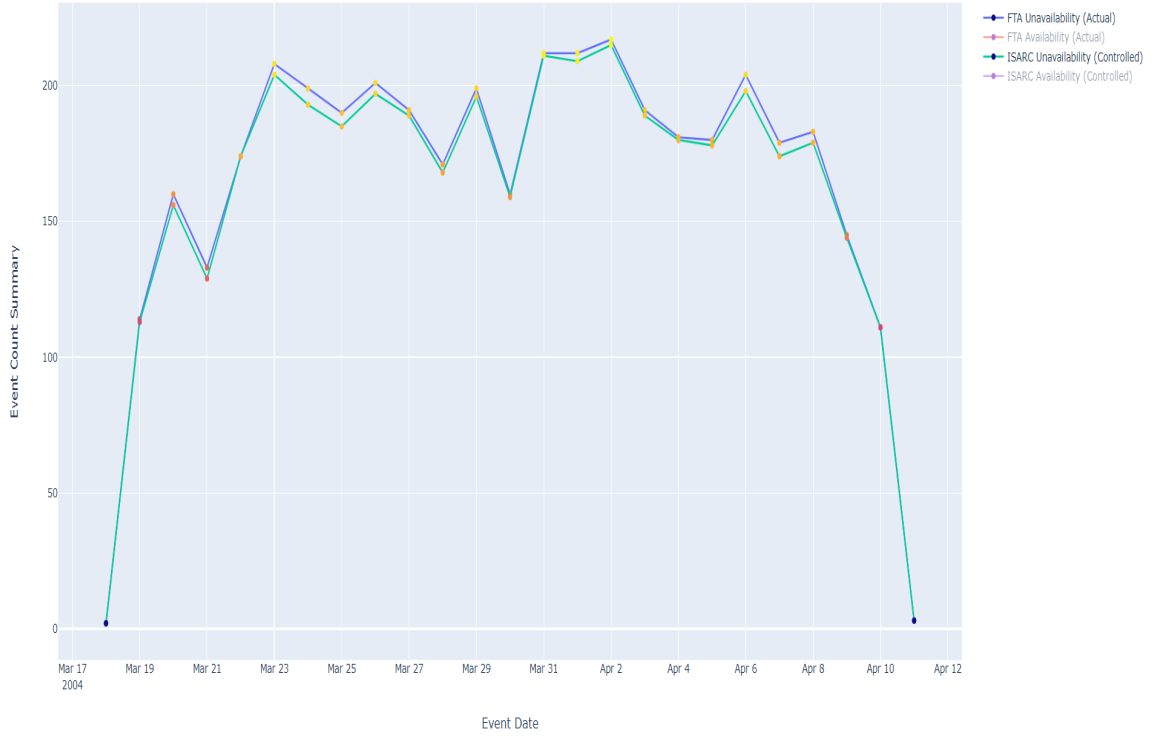


Figure 5.19: A modified PI Control Implementation for the ISARC using Lundberg (2018)

Table 5.18: ISARC Decision Model Impact for PI Variation

Kp(Row) $\Rightarrow$ Ki(Column)	0.0	0.5	1.0	1.5	2.0	2.5	3.0
Configuration 1	0.0	z	z	z	z	z	z
Configuration 2	0.5	z	z	z	v	v	v
Configuration 3	1.0	v	v	v	v	v	v
Configuration 4	1.5	v	v	v	v	v	v
Configuration 5	2.0	v	v	v	v	v	v
Configuration 6	2.5	v	v	v	v	v	v
Configuration 7	3.0	v	v	v	v	v	v

states to the high interaction nodes. For 5000 samples, the findings produce an output that maintains a consistent shape. All configurations yield a similar shape when compared to Figure 5.19. The results are repeatable using a combination of PI control ranges. For the average corrected control count over the measurement period for each configuration of Kp and Ki;

1. the `np.mean(ISARCResultSetBSimulationComparison_AppliedControl['FTA Unavailability Actual'] - ISARCResultSetBSimulationComparison_AppliedControl['ISARC Unavailability Controlled'])` applies to the matrix in Table 5.18. The result of each configuration yields a single row and column combination of proportional (Kp) and integral (Ki) gain.

Table 5.18 produces a consistent range of values from 0 impact to a maximum of 1.84 for the

5000-sample run. However, the linearity of the control actions limits up to the 5000-sample set. Increases in sample sizes are a suggestion for future work. One certainty is that for $K_p \leq 0.5$ and $K_i \leq 1$, no additional control manifests for the 5000-sample set. These configuration types reflect a z (zero). Additionally, for configurations where $K_p = 0$ and $K_i \leq 3$, no additional control is possible in cases of the 5000-sample control sample. Matrix values v (variable) indicate the variable response boundaries for the simulation in Figure 5.19. Table 5.18 produces a point in time configuration matrix for determining control impact on the subset of FTA measurement data. The data set tests against the fixed FTA measurement set and achieves a single peak value. Additional peaks are possible with larger populations. The code blocks in the body of work of this thesis support the ongoing configuration of decisions that will hopefully draw interest from Systems Architects, Systems Planners, and stakeholders that aim to explore architectural decisions for the Internet of Things.

Finally, Figure 5.20 indicates the results overview in the context of the final results for the thesis.

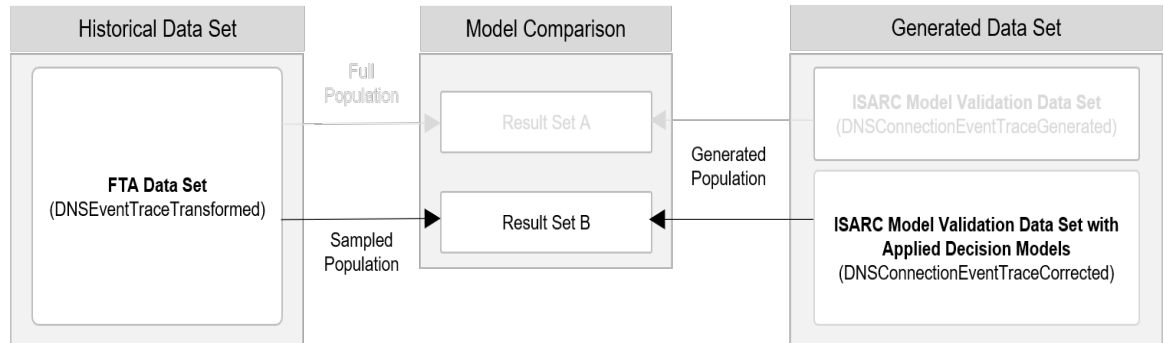


Figure 5.20: Results Summary B Overview

5.11 Chapter Summary

Chapter 5 deals with two validation types for the ISARC. The first model validation approach deals with model extension to the primary OPM model in Chapter 4. The extensions exhibit flexibility while maintaining model integrity. The extended OPM model integrates device identification by modelling device address assignment as a precondition to device identification. Additionally, Chapter 5 produces device discovery, including handling state errors during the device discovery process. Device managing produces the final in-system test of an interrupt against the known Android constant signalling. The model extension finalises with system hierarchy placement arguments. Finally, the extension summary argues for the IoT System Controller's attachment to serve a control purpose at different layers in

the ISARC system hierarchy. The OPM-specific validation section completes with OPCAT proof of model validation between two different control initiation points.

Chapter 5 also details the handling of Decision Model DSMs, the ISARC **Front-end System Set** data generator, and the static FTA DNS data set. Results show that the ISARC **Front-end System Set** data generator produces comparable models with the FTA concerning random node matching sets. The initial sample set ranges of 10000 to 384991 indicates that random matching for the same LDNS node ranges produces an error margin of -0.3117 to 0.3117 compared to the static {1:Availability} and {0:Unavailability} connection state split of 58% and 42% found in the FTA data set. A significant finding is that without a hard-linked data generation approach, the **Front-end System Set** data generator adheres to the characteristics of the LDNS data set. Chapter 5 finally applies the ISARC decision model to provide a mechanism to handle different configuration profiles to tune Failed Connection Percentage (FCP) and Successful Connection Percentage (SCP) within reasonable margins that exist in the **Front-end System Set**-FTA data sets. Zero and variable change indicators on the prioritised controller implementation yield a series of configuration options to enable a shift of margin values towards the fixed {0:Unavailability} and {1:Availability} statistic in the FTA data set.

Chapter 5 solidifies the view on handling quantitative architectural decision variations for the IoT-Ce. Preceding chapters have discussed vertical themes which culminated in the creation of the definition-to-OPM approach - a parameter extraction process based on principles by Dori (2002), ISO/TC 184/SC 5 (E) (2014) and Crawley et al. (2016). The IoT requirements extraction process presents a composable set of models that contain singular objects enabling hierarchy-based placement. Using Python, a codified Design Structure Matrix (DSM) presents a compact and representative view of the decision model against which the evidence validates. A Python code block applies a concrete control logic for handling proportional and integral control in cases where the state of an **Front-end System Set** needs to change to meet availability requirements exhibited in the FTA DNS data set. The DSM intersection values produce unique interaction sensitivities that support increased decision purpose through input column notations that further enable clustering of extracted parameter values.

Chapter 6

Conclusion

The evolution of edge systems has reached a point where these systems touch our lives in a unimaginable ways. These edge systems have become the physical extensions of the internet. This cyber-physical evolution goes beyond the exposed services from individual embedded devices. It underpins the Internet of Things (IoT) by creating a connected edge (Ce) system that integrates large-scale digital and physical processes. Due to this growth trajectory, design methods closely aligned with the natural order of systems design are necessary. A need exists to understand the arrangement of components and their interactions that exist in the IoT-Ce. This need gave rise to;

1. propose a standards-based systems architecture that outlines the runtime configuration of IoT application components found in the IoT-Ce, and
2. a model framework that enables the repeatable generation of data sets using the model's output to allow model tuning via a novel control strategy in line with system dependability impacts. The IoT-Ce system purpose models an abstract value-based statement that attaches to lower-level process interactions using a single notation.

In Chapters 4 and 5, OPM was used to model, test, and continuously define the notion of IoT-Ce systems, subsystem and component architectures. The placement of these components into a natural systems hierarchy was exhibited. The outcomes of the continuous analysis of the placement of components deliver results that form the basis of the object process diagrams (OPD) and self-generated object process language (OPL). The use of constant learning, dual-channel outputs, and creating appropriate boundaries limits the content characteristics in the ISARC. OPM applies Mayer's Cognitive Assumptions, as indicated in Dori (2011), by using native processes decomposition through zooming. Additionally, the decomposition of form occurs through structural folding and unfolding. The natural use of model symbols in partial OPM conformance further binds the OPM's applicability for the

IoT-Ce. The use of definition extraction from the IoT body of work shows the transformation of basic object, process, and state possibilities to a concrete decision space, decision segmentation, and metric attachment to guide the outcome of the decision process. Crawley et al. (2016)(pp.326-327) indicates these heuristics for creating architecture decisions. The answer to the research questions deliver a view on;

1. How the runtime configuration of IoT system, subsystems, and components support dependability through reliable state awareness of the IoT-Ce. It is primarily in understanding that object, process, and state enable the hierarchy in IoT-Ce systems architecture. To guide certitude in the modelling of the ISARC, the general principles of OPM through appropriate symbol usage, hierarchical placement, and pre-existence are necessary. The main systems architecture realisation concepts identify as *stating the problem occurrence, identifying system purpose, understanding the primary function above other secondary and supporting functions, model the human and non-human aspects of the system, and place objects and process within or outside of the system boundary*. These rank as the correct drivers for modelling robust IoT-Ce systems architecture. The OPM further determines state indicators that bridge the gap between perceived and experienced system behaviour. Access to high fidelity and credible models enables decreases in uncertainty in systems. Additionally, the handling of both visual and textual model outputs reduces model inconsistency.
2. How to model different architectural options in a heterogeneous system to drive minimised error and control actions for the IoT-Ce? By inheriting a static view of past system behaviour, the ISARC model output generates a data set that mimics the fixed state ratios of a historical data set for a domain name system. The significance of the historical data set is that heterogeneous node structures abstract into a request/response interaction toward active and failed states in the local DNS measurement set. The edge system configuration produces randomised, instead of fixed, node access attempts. The model outputs delivered in the contribution is repeatable in a margin range of **(-0.311676 to 0.311676)**. This range represents the difference between the generated ISARC data set and the fixed FTA measurement set. The margin differences derive from the fixed FCP and SCP values shown throughout the handling of the FTA data set - DNSEventTraceTransformed - where the FTA Failed Connection Percentage (FCP) target is *41.9218* and the FTA Success Connection Percentage (SCP) target is *58.0782*.

Given the hierarchical nature of the decision model outputs in ISARC Decision Model 1 through 4, and the repeatable and nature of the simulation sets in Code Block 5, the claim that;

“Low level IoT-Ce configurations that influence decisions about dependability can be abstracted into a conceptual IoT system architecture in order to support model validation activities” is seen as valid in the case of a comparison of a generated data set against a fixed data set with a fixed systems hierarchy.

In this thesis, a complexity-reducing modelling method models the varying nature of IoT-Ce configurations with the notion of system state at its very center. This recommendation is based on the case that a simple set of interactions for the IoT-Ce appears to fit into OPM’s use for representing the systems architecture. Additionally, the system model links high-level system value to lower-level process interaction states, such as those seen in standard Android OS libraries, hence reflecting the configuration on an actual IoT-Ce **Front-end System Set**. The hierarchy above manifests using a single modelling notation focusing on objects, processes, and state changes. In addition to more straightforward model representation and creating a series of structural system boundaries, the ISARC, therefore, recommends using OPM ISO-19450. The inherent function and form outcomes of using OPM indicate that core systems architecture questions are answered when dealing with a complex system such as the IoT-Ce. By its very nature, the IoT-Ce is a bounded system in that the Ce is concerned with components and subsystems that exhibit end-user control as IoT-Ce **Front-end System Set** devices such as sensor-rich mobile phones. These devices exist within the boundary of control away from Cloud or far-end system sets. Dori (2019b) states that five keys system realisation concepts relate to the modelling of a systems architecture. Using OPM ISO-19450;

- The **Problem Occurrence** outlines the overall system parameter that is in jeopardy if the system does not meet its primary intended purpose.
- The **System Purpose** outlines the main reason for the system’s existence. Namely, the value that the system produces. This value appears as the top-level node in the OPM Systems Diagram (SD).
- The **Primary Function** (Process and Object Transformation, or Operand Transformation) appears as a value delivering process that contains sub-processes. For the IoT-Ce, it is the ability of the **Front-end System Set** to acquire and generate sensor measurements, to transmit measurements, and to analyse measurement data.

- The **Enablers** of the system such as agents or instruments represent human or digital components that operate the system. The **Front-end System Set** is an instrument in the value statement for the IoT-Ce.
- The **Environmental Factors** represent factors outside of the control of the system, such as impacts of variable protocol selection on transmission time, data structure sizes, communication requirements for specific **Front-end System Set** nodes, and so forth. Several factors present plausible arguments for systemic instead of environmental placement. The process of defining elements that serve to fall within system control, and those that do not, increases the scope of system modelling resulting in the inability to deliver on the primary system value.

The OPM standard recommends an object and process view of the system, that when discussed together, enable state change within the system boundary. Coupled with the OPM, Design Structure Matrices (DSMs) structure interactions of system, subsystems, and components in a compact form. In designing complex systems, these two methods enable the specification of systems far more complicated than IoT-Ce systems. This thesis suggests that these methods apply to produce system models for the IoT-Ce, but more so for general IoT systems design to reduce specification inconsistencies. Additionally, the use of these methods enables parameter-based decisioning for the systems architecture. Parameter-based decisioning methods offer a means to create architectural decision trade-offs for supporting sound architectural decisions. Parameter-based decisioning supports simulation of multiple decision parameters, producing varying decision combinations that translate into a programmable, repeatable simulation set. The ISARC model uses Python to simulate the structural integrity of the system by comparing the model output to a fixed data set. The ISARC model proof aims to provide a more extensive sampling population compared to visually checking the model in OPCAT or OPCloud. The Python model proof offers an extension to modelling interaction probability and simulates these under different IoT-Ce configurations.

Based on current knowledge, the handling of OPM parameter-extracted systems architecture and DSM for the IoT-Ce is not explored in the IoT body of work. Additionally, the applied Python generation approach for an OPM-extracted parameter-based decision model against fixed historical data sets has also not been undertaken at the time of publishing this thesis.

6.1 Thesis Contributions

Two contributions are present in the thesis. The study proposes a standards-based systems architecture to outline the systems, subsystems, and components employed at runtime in IoT-Ce systems. The systems architecture applies the Object Process Methodology. The **IoT Systems Architecture for Runtime Configuration (ISARC)** models a hierarchical structure for a multi-layered connected heterogeneous IoT edge system. The study of the runtime properties of systems proposes a sturdier understanding of how to model dependable and scalable systems. Additionally, these systems have the requisite properties to find application in large-scale distributed systems design. This contribution's primary benefit is a proposed approach to model an abstract to concrete systems architecture using a single notation that applies complexity reduction measures. The ISARC produces a well-defined set of system, subsystem, and component boundaries. Assertions on hierarchical placement of subsystems and components are also explored in this thesis. The proposed approach simulates the model against a known historical data set created using a well-known two-tiered client-server architecture - a domain name system (DNS).

Secondly, the thesis provides a simulation that outputs a data set compared against a failure trace data set for a DNS. The study's uniqueness is that the ISARC data set is compared directly to the FTA DNS data set to suggest that changes in interaction sensitivity in the ISARC can provide a marked improvement level when generating random samples from both fixed and variable data sets. The FTA for DNS is known to contain measurements over a period in March and April 2004 where 62201 IP-based nodes produced an availability and unavailability state data set containing 384991 events. The thesis could replace the data set could with several other data sets in the FTA. However, the data set's likeness in terms of the hierarchical node structure makes the FTA data set amenable to **Front-end System Set** structure argued in this thesis. Once produced, the ISARC simulates interactions in the structural hierarchy using a novel random approach. Incremental decisions related to interactions in the IoT-Ce apply a higher sensitivity, whereas components that are less likely to interact are assigned a lower sensitivity. For interacting components, a concrete control constraint suggests that the structural model for the ISARC can generate a dataset that comes within reach of the Failure Trace Data Set for DNS. Thus, the study claims that modelling the IoT-Ce systems architecture using OPM, DSM, and applying a novel control strategy; the ISARC can generate a data set that closely mimics a historical point in time. One could argue that a randomly generated data set with constraints can generate the same FTA measurement set. A warrant that makes this claim reasonable is the flexibility in data

generation tools today. However, what is not always considered in general-purpose data generation libraries and tools is the simplified interaction between object, process, and state within the modelled system. Furthermore, model-based generation and learning tools can focus on supervised and unsupervised training and could thus be applied. However, it has not been tested against the relationships between OPM, DSM, and the IoT-Ce.

Recently, the OPM and DSM have found their way into industries outside of aerospace and manufacturing. The proposed findings do not aim to re-establish individual vertical areas of interest in systems design but rather to succinctly position a verifiable hypothesis by creating a standards-based model for proposing decisions on runtime configurations for the IoT-Ce. The thesis results present suggestions that;

1. OPM enables the scaling of a **complex system down to single domain**. It creates a single purpose system boundary, namely the IoT scaled down to the IoT-Ce,
2. the IoT-Ce **systems architecture** can conform to partial OPM ISO-19450 use. The diagrammatic representation of models found in Chapters 4 and 5 present specific system diagrams as [SD IoT System Value](#), [SD1 IoT Edge System Value Delivering](#), [SD1.1 IoT Data Transmitting](#), [SD1 Extension for System Value Delivering with Environmental Control Configuration](#), [SD1.1 IoT Data Transmitting with Environmental Control Candidate](#) and an unfolded structure for an [IoT System Controller Candidate](#). Additionally, Chapter 5 contains further decomposed model sets for [System Request Handling](#) showing, [Device Addressing](#), and [Device Discovering](#). These processes indicate to support the value delivering processes in the IoT-Ce,
3. a **series of Python data generation code blocks** can produce a data set within relative distance to a fixed measurement set for a multi-node systems hierarchy,
4. a **series of applied control constraints** brings the generated data set within a reasonable error range to the fixed measurement set, and
5. a **configurable Python code base** that houses structural decision tradeoffs propose a method for ongoing configuration decisions for the IoT-Ce.

The model successfully achieves a simple error margin of -0.3117 to 0.3117 , and a suitably controlled error margin, respectively. These margins apply to the fixed FTA Failed Connection Percentage (FCP) target of 41.9218 and the FTA Success Connection Percentage (SCP) target of 58.0782 found in the FTA data set.

The model output in Figure 6.1 presents an overview of the thesis contributions. Where ISARC Decision Models 1 to 4 produce the structured inputs for the DSM code blocks, code block 5 produces the handling of the master FTA data set. Result set A and B outputs yield from the code block that generates the ISARC model validation data set. These output data sets compare to the master FTA data set, yielding the margin results mentioned before.

In conclusion, the model proof indicates that a change to the control aspects of a two-

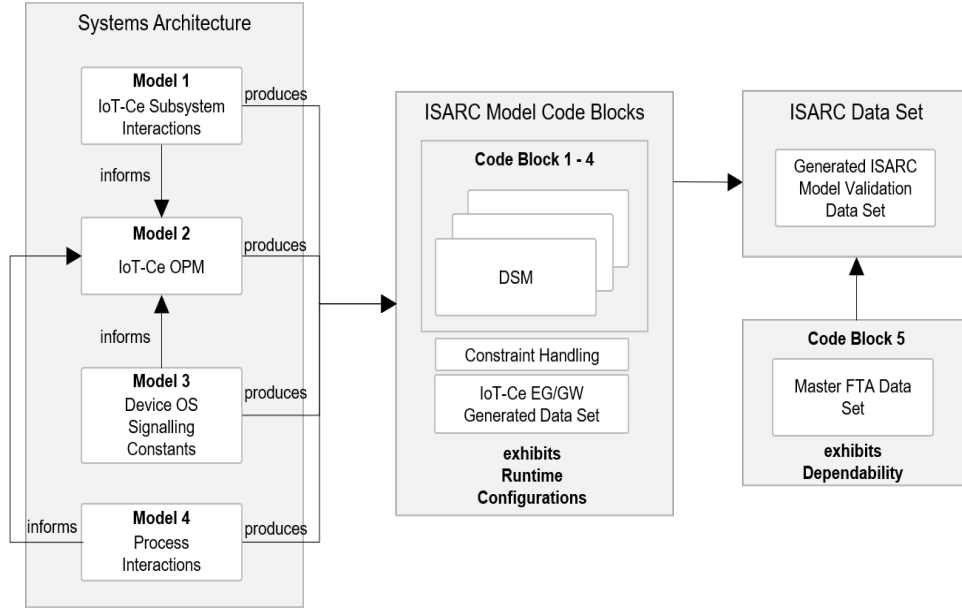


Figure 6.1: Integrated Thesis Contribution

tiered systems architecture yields a positive result in attaining a simulated value for levels of dependability in an **Front-end System Set** architecture. This claim bases itself on the model generating a data set, that when sampled against a fixed measurement dataset, is within an acceptable error range between availability and unavailability states. The simple comparison sample set uses sample sizes of 10000, 20000, 50000, and further towards the entire population of 384991 events.

The random output sample tests are executed more than 100000 times, which exhibits a normal distribution for a standard deviation in the controlled result. These tests apply considerations for connectedness in the IoT-Ce systems architecture, where the type of interaction conforms to a high degree of decision sensitivity. By understanding the diagrammatic systems architecture, one can model the form, function, and associated state changes that provide a broader understanding of the handling of processes within the system boundary. Additionally, by attaching sensitivity scales to the known part of the system, a simulated

view of ongoing decisions about the system can produce results that lessen error margins for determinism in the system.

6.2 Further Work

The future of lightweight rich sensor and embedded technologies that enable user-specific utility is evident in the vast research body for embedded systems, cyber-physical systems, and IoT. The technical integration of these technologies closes the gap between physical and cyber requirements. The co-design of systems that serve immediate satisfaction of requirements is evident in embedded systems, cyber-physical systems, and IoT design. As the need for technological advancement reaches unimaginable proportions, these technologies expand human needs and demand greater stakeholder value. Strategies to naturally verbalise the interaction between systems, subsystems, and components for the IoT will be needed as greater complexity finds its way into systems architecture. Strategies to control **Things** to yield increasing reliability through self-healing and adaptation is already at hand. In the same way that mobile devices seamlessly connect to mobile networks, more efficient ways of presenting reliable services in the IoT-Ce are undoubtedly required. However, before considering this large influx of multi-domain impacts, consider the future as one that requires specification of systems using more straightforward systems design methods. Methods that; reduce complex model transformations, reduce ambiguity present in multifaceted specification documents, reduce long model-to-test cycles, and clarifies incompatible semantic communications.

The OPM ISO-19450 is a systems automation and integration standard that only complies as far as human interaction and knowledge is applied - as in the case of the ISARC. OPM creates a single notation possibility that reduces the semantic specification distance between diagrams. The output creates individual parameters that serve to describe system problems in their most compact form. However, even though the OPM creates a natural structure for the system's evolution, it also requires an additional framework to place parameters such that it can simulate extensive component interactions. This additional framework is where Design Structure Matrices provide compact structural design specifications. DSMs have many uses, including specification of simple and complex products, processes, and organisational interactions. Additionally, DSMs offer a means to automate subsystem and component clustering using programmatic functions instead of declarative functions - as seen in this thesis. Finally, the decision model outputs from this thesis base themselves on the process activities specified in configuration automation. Given the underlying principles that the ISARC builds on, future work can consider specifying additional underlying system

configuration and their impact on other self-* properties of a system. The use of ISARC can extend to use cases that require the conceptualisation of rapid adaptive control specification. Additional implementation and modifications to the ISARC can apply to;

1. **increase coverage within the single system boundary.** Obtaining greater depth for compositions that might stretch across geographical boundaries. The system's complexity in this thesis is bound by considering that devices at the edge produce generation, acquisition, transmission, and analytical functions. This functional view is due to the influx of changes to mobile devices and the value that each device can create outside of voice communications. Additional model constraints could further address the shrinking form of components found in the IoT-Ce. The current thesis addresses the ability to address and identify **Front-end System Sets**. It suggests that applying control to these functions can impact the overall system value they provide in connection reliability. Further work on boundary selection suggests the behaviours that relate to **Front-end System Sets** can decompose into different subsystems and component function and form. An example of this is the inclusion of packet transmission time, data structure sizes, and power requirements for specific lossy ED and richer GW nodes. Effectively changing the affiliation of these parameters to systemic as opposed to environmental. Additionally, the inclusion of human-centric processes that form the basis of configuration management, integrates both **instruments** and **agents**. Further work in the area of human-centric function and form manifests in the use of Multidomain Architecture MDM Models as indicated in Eppinger and Browning (2012)(pp.237). The flexibility of the method enables these to be included within OPDs or directly into DSMs.
2. **a different perspective on the use of the OPM ISO-19450 systems architecture outputs.** The adaptation of objects and processes for automated control using natural language processing can offer a remarkable extension to the ISARC, or a derivative. One of OPM's key characteristics is dual-channel processing, namely text and graphical output via a single model notation. Thus, creating a Python library that consumes the standard text outputs of the model offers possible increases in the core OPM standard to drive toolmakers conformance. Three levels of ISO conformance exist for OPM, namely conformance at a symbolic level, full conformance, and conformance for toolmakers. The ISARC applies symbolic conformance. Even with the minimal conformance profile, the models still produce text hooks such as *entity essence for agents and instruments, handles, consists of, exhibits, can be, is, requires,*

and changes. These are primarily structural and interaction hooks that can undoubtedly serve to train natural language processors. Haapala (2014) produces a Python library that implements the Levenshtein Distance, which aids in calculating the similarity of text descriptions using fuzzy methods. For architectural decisions that contain textual descriptions for the system, subsystem, state, and systems architecture form and function, the matching of decisions can yield clarity where the ambiguity of the parameter description is under question. For example, the impact of communication decisions on hardware selection between GW and lossy ED components in a connected edge subsystem.

3. **modification of the Python code blocks** that integrate the Python decision model with a standard IP-based web service component set for enabling control decisions in the IoT-Ce. In the current thesis, part of the DSM structure (ISARC Model 3) exhibits an abstracted Android OS library. This library provides access to operating system signalling and control. The literature reviews on CPS and hardware/software co-design give rise to the references in the thesis on bridging the gap between hardware and software specifications. The suggestion in the thesis is that an understanding of the arrangement of parameters in the system provides the basis of decisions before adaptive design or reconfiguration. Although used as a primary reason for the ISARC hierarchy, representation of CPS parameters and their impact on dynamic behaviour in the IoT-Ce should be undertaken.
4. **extensions to the control theory used in the ISARC**. The control constraints that produce increases in accuracy against the FTA data set are limited to proportional and integral control. The primary concern in the thesis looked at applying control to change state values. The Python code blocks handle the control action against an immediate event and integrated control action. Further work can be considered in managing subsequent event controls errors, such as the principle of derivative control. Coupled with the text processing model suggested for OPM, an object-process combination that will handle state in a future event could increase the accuracy of the simulated data set. Further work should integrate additional PID principles to change the interaction probability stated in the required DSMs. PID control overviews appear in Appendix A.3.1 and the python code blocks are agreeable to change.
5. **additions to the configurable code base** create a decision model to house structural decision tradeoffs by reconfiguration. In industry today, the MAPE-K forms the basis of adaptive systems design. The literature review has shown that MAPE-K variations are still underpinned by the primary processes of visibility handling, state

analysing, planning, predicting, and executing. Thus, further work in MAPE-K expansion by codifying configuration decision sensitivities into the MAPE-K enables automation in the decisioning phase, and knowledge repositories. Structural decision models with variable domain inputs such as Cloud Computing for IoT, Industrial IoT Topology Decisions, and automated decision constraint handling for global IoT deployments are exciting areas of focus in IoT ecosystems. These focus areas are primarily due to physical and logical system boundaries that enforce interaction constraints across systems, subsystems, and components.

6.3 Closing Statements

The study proposes a method that considers simplifying the specification of an IoT-Ce system from something abstract such as stakeholder value, to something concrete, such as a process change in smart mobile devices. Ordinarily, the hierarchy of a system splits into different model types, even in a single standard such as UML or SysML. The OPM ISO-19450, coupled with DSM, reduces the need to transform models while reducing ambiguity in specifying decisions for the IoT Connected Edge. The ISARC shows that a large complex system will consider those decisions that can impact its integrity. In addition to identifying the OPM and DSM as a suitable systems model combination for the ISARC, the extraction of the system purpose of the IoT-Ce components from the definitions of IoT in an existing IoT body of work uses standard OPM Object-Process-State structures and symbols. This method is the **definition-to-OPM** approach.

The proof that the model hierarchy is structurally sound is shown by generating a data set and enforcing a series of system constraints so that the production of the data set closely matches that of a historical data set. The thesis recommends the OPM/DSM-based systems architecture approach utilising the following structure;

- **Chapter 1:** Creates an overview of the primary themes considered in the main arguments.
- **Chapter 2:** Creates the theoretical basis and fundamentals of Systems Architecture, IoT, System Configuration, and Architecture Decisions.
- **Chapter 3:** Creates an overview of five core Systems Architecture questions when considering OPM as the primary systems modelling method; *problem occurrence, system purpose, primary function, human and system enablement, and environmental factors*.

- **Chapter 4:** Applies the system principles to the system value by showing conformance to OPM symbol usage for the structural and behavioural aspects of IoT-Ce systems that drive configuration variability. Additionally, Chapter 4 demonstrates the main system purpose and primary system interactions, supporting objects and process transformations, system interfacing, interactions that support control model types, and process variation and configuration.
- **Chapter 5:** Extends the primary use cases by maintaining the ISARC SD hierarchy and arguing for extension on additional testable functions; namely [Device Identifying](#) with an [Addressing](#) pre-condition, [Device Discovering](#) and [Device Managing](#) that applies a hard operating system interrupt. Additionally, Chapter 5 produces a fixed output dataset that compares to a historical measurement set for hierarchical system reliability.
- **Chapter 6:** Provides summarised remarks to conclude the findings and recommends further work.

Appendix A

Additional System Considerations

A.1 Additional Considerations

A.1.1 Dependability Characteristics

With IoT systems becoming more complicated due to variation in standards and vertical solutions, they will need to become increasingly flexible while maintaining suitable reliability levels in a physical behavioural context that remains primarily random. An example of such behaviour is executing a control action in response to a latent sensor measurement response in critical applications - such as an intelligent traffic light system or an intelligent gateway system. The resultant control action based on a temporally latent measurement could be catastrophic. This action follows a “hard constraint” - a term that this study adapts from the known definition of hard time constraints by Kopetz referenced in Marwedel (2010). The term “soft time constraint” is further adapted as “soft constraint” to argue for reliable system behaviour in a changing heterogeneous ecosystem for consumer devices. Additionally, a median sensitivity constraint that is neither hard nor soft could argue towards systems that support use cases that focus on individual human-centric system value drivers such as those seen in smart phone-based IoT.

A physical-to-informational design perspective within interconnected systems considers basic concepts such as fault management, control, filtering, and identification, as seen in Aubrun et al. (2010)(p.10). These are amongst the most well-known concepts that drive dependable systems design. Furthermore, the detection of faults that applies to automated systems control focuses on maintaining certain states at their set-point value and is also concerned at a system product level about the system and variations in behavioural aspects to handle faults efficiently shown in Aubrun et al. (2010)(p.3). The basic dependability characteristic which incorporates a definition of reliability as seen in Marwedel (2010) and Aubrun et al. (2010)(p.3).

However, the challenge that raises concerns is the integration of next-generation technologies into older design approaches. The core consideration in realising physical-to-informational design places itself in integrating embedded and communication technologies indicated in Marwedel (2010) and Kopetz (2011). This resolution of this challenge outlines the role of embedded systems as “disappearing” technologies, Marwedel (2010). Additionally, the measured effectiveness and efficiency for this integration manifest in embedded systems design characteristics such as dependability, which incorporates measures of reliability Marwedel (2010) and Kopetz (2011). This characteristic binds via principle design differences between general systems and embedded systems, Henzinger and Sifakis (2007) and Marwedel (2010). Where embedded systems design follows a somewhat complex software-hardware structure as seen in Figure 2.4, the requirements between hard and soft constraints generally appear in critical and non-critical systems Kopetz (2011), respectively. The key difference between design challenges is the tradeoff considerations on the impact of a breach of constraint boundaries. A breach leading to loss of life such as those found in Rammig et al. (2009), would require stricter control. In contrast, a violation within a more comprehensive acceptable range could mean control adjustment according to an agreed quality of service. Partitioning and boundary principles are thus required to enforce limitations on operating conditions for systems. These principles are well known in system design, Eman et al. (2002), based on the ability of the system components to exhibit cohesion and coupling principles. Therefore, system boundaries reinforce an ‘error containment principle’ which implies that the system should partition to contain cascading errors, Kopetz (2008).

A performance analysis model in embedded systems provides a baseline for protection from cascading errors in a system hierarchy. This baseline exists due to design considerations for embedded systems performance, which underpins questions relating to partitioning, allocation, and binding, as shown in Thiele and Wandeler (2005). The partitioning component contains the decisions needed to ensure that software and hardware conform to the previously mentioned cohesion and coupling principles. The requirements outlined in Thiele and Wandeler (2005) provides further evidence that a systems engineering approach to stipulating form, function, and interaction is necessary. Specifying the architecture to support variations through decisions, configuration, and function-to-form mapping reduces analysis time for several architecture decisions and architectural constraint tradeoffs. Furthermore, modelling the correctness of a system ensures no state that the system cannot achieve, as shown in Thiele and Wandeler (2005). Thus, the design of a system under a modelling regime will contain a model based on characteristics such as modularity, cohesion, coupling,

composition, and aggregation. Using composable systems design for protection strategies meet the core requirements for efficient boundary setting. Thus, controlled non-functional system requirements focus on dependability and reliability.

A.1.2 Reliability Constructs

In defining the structural and functional components of an IoT ecosystem, making specific reference to the runtime arrangement of this architecture is necessary. Thus, the configuration at a point in time, either negatively or positively, impacts the system's reliability. The question is, How? As the architecture varies at runtime, different items (components or even entire subsystems) may attach or detach from the ecosystem. For example, connecting a new smart sensor or actuator component within the edge subsystem at runtime could result in a failed connection. This configuration variation further extends to uses cases where gateway power failures may occur after the first operation, resulting in gateways unable to send or receive appropriate control commands. For both use cases, let us imagine that the event occurs in the past. However, the state of the IoT-Ce component is currently operational. Given the state variable $X(t)$, the state for the functioning component would be 1. It will be 0 if the component is in a failed state. As indicated in Høyland and Rausand (2009), the state variable appears as;

$$X(t) = \begin{cases} 1 & \text{if the item is functioning at time } t \\ 0 & \text{if the item is in a failed state at time } t. \end{cases} \quad (\text{A.1})$$

Thus, it follows that the next time a failure occurs, variable $X(t)$ would be 0. Additionally, this state change would be random in nature Høyland and Rausand (2009). The basic reliability - survivorship - function is represented as the probability (P) that the component is functioning at time t . This probability presents as;

$$R(t) = P[T > t] \quad (\text{A.2})$$

T represents the component lifespan, whereas **t** represents the point in time for the known state of the component. The state is namely functioning or not-functioning. Therefore, the reliability at the time (t) is the probability that the components are operational at time t with time period T. In reduction towards a steady-state, a series of definitions for faulted, failed, and error states are required to guide the functional boundary of the state simplistically, Høyland and Rausand (2009). However, understanding all failure modes is difficult in any system, as indicated in Høyland and Rausand (2009), but more so an integrated heterogeneous system. Figure A.1 shows the transition of these concepts from those that can behave within acceptable limits (errors and failures) to those that require a more significant

deal of interaction to resolve (faulted). Standard probabilities and operating reference data across digital communications domains, reliability, and mobile communications appear in comprehensive engineering reference data collections. An example of reliability and testing data appears in references such as Marriott (2002). Although not entire applicable to the argument, note that a guided and standardised measure of system reliability and operating conditions are available.

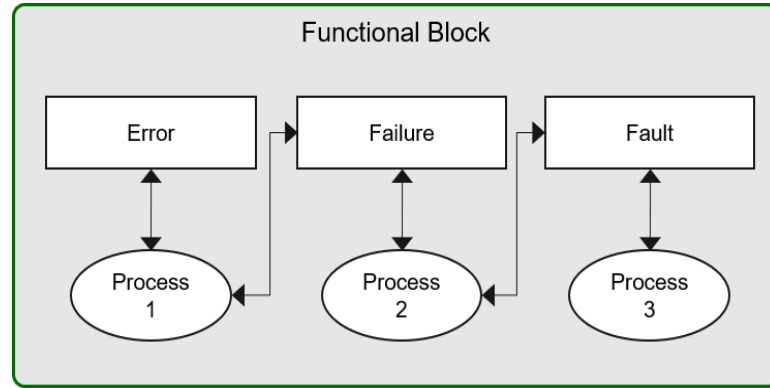


Figure A.1: An Error State Escalation Flow adapted from Leon-Garcia (2008)

Thus, the definitions of error, failure, and faulted states position as possible aggregate states within the IoT system boundary.

A.1.3 Random System Behaviour

In systems, random behaviour can cause the system and associated parts to produce models with incorrect assumptions. However, modelling with accurate real-world constraints in mind increases model fidelity. Fidelity is a systems engineering characteristic that increases the level of trust that the model will behave similarly to real-world scenarios. Several model characteristics in system engineering texts exhibit a trade-off between trust, cost, and benefits of the modelled system. Texts such as those used to frame complex product design, Crawley et al. (2016) and Cameron et al. (2017), provides considerations for guiding principles within which system trade-off occurs.

The IoT contains a series of randomly connected devices with random states. This viewpoint aligns with random placement behaviour seen in general IoT use contexts. These are similar to random node placement in wireless sensor networks, Bera et al. (2016). Under this definition, randomness occurs as a numerical attribute, namely a state at a point in time, in a finite discrete measurement period. Furthermore, the understanding of the state at a point in time deduces that the extraction of a random sample will assign a numerical value to the

measurement. This view is stated as the standard definition of a random variable, Leon-Garcia (2008). A sample space for vital state transitions can, therefore, occur using zero (0) and one (1) given the definition of failed versus functional components as seen in Høyland and Rausand (2009). Extending a coin toss example seen in Leon-Garcia (2008)(pp.57), novel rules that produce a 10-measurement set of 5 measurements containing a choice of zero (0) or one (1) applies. The sample space (S), therefore, presents the outcome Z in the table below. Due to X taking up a value in the set $\{1,2,3,5\}$, the set produces a series of

Table A.1: Random Sample Logic adapted from a coin toss example in Leon-Garcia (2008)(pp.97)

Z	01101	10101	11111	01100	00111	11001	00100	11000	10000	11111
X(Z)	3	3	5	2	3	3	1	2	1	5

finite-state selection options. Thus the probability can be calculated using a Probability Mass Function (PMF) represented as having unique instances of set occurrences found in the sample space (S). In the example above, the number of measurements is limited to five

Table A.2: Probability Mass Function Summary adapted from Leon-Garcia (2008)(pp.101 & 102)

Set Value	Occurrences	Total Measurement
1	2	10
2	2	10
3	4	10
5	2	10

per measurement set. For example, five sequential state reads spaced in 1-second intervals for a single IoT device. This example expands to multiple sequential-state reads spaced at a variable time step for a large number of IoT devices. Gateways and physical space drive the amount of devices that one can comfortably place while maintaining some form of value from you achieve from an IoT device. The further extrapolation of this example exhibits the massive randomness when the problem space contains a multi-parameter configurations. Fortunately, the random constructs execute by creating a finite problem space containing a finite number of states, IoT devices, and connection parameters. Furthermore, additional constraint parameters bind the problem space into a finite domain. Although the state domain is limited to state zero and one, standards such as the process constant arrays found in the IoT/CPS application domain in Linux (2018), and those found in mobile network configuration standards in TC-SMG (1996), provide a broader set of configuration states against which functional and non-functional states maps. Additionally, the transition of known process states seen in Reichert and Weber (2012)(pp.34) provide a limited series of process interactions.

Finally, system engineering guidelines such as those found in Crawley et al. (2016), and Cameron et al. (2017) offer guidance for dealing with systems, subsystems, components, and the interfaces that connect system hierarchies. The nature of these systems are centralised or distributed, and thus Crawley et al. (2016), and Cameron et al. (2017) offer a means to provide architectural guidance for binding runaway system behaviour. Therefore, it provides the necessary system boundary principles within which modelling of random system behaviour can occur.

A.1.4 Failure Analysis in Distributed Systems

Suppose that a system consists of several components or subsystems - the reliability of this system can be computed in terms of the reliability of its constituent parts. This assumption holds if the components that support such a subsystem fail independently of each other, Leon-Garcia (2008).

In Table A.1, the first measurement set is represented as 01101. This combination effectively means that the first measurement showed `State::Failed`, the next reading shows `State::Functional`, the following `State::Functional`, and so forth for the entire sample space. This initial view concurs with general random system behaviour as indicated in Høyland and Rausand (2009), where the time to fail occurs randomly at some point. In the case of the first reading, the system fails immediately at reading 1. The curve, which represents a

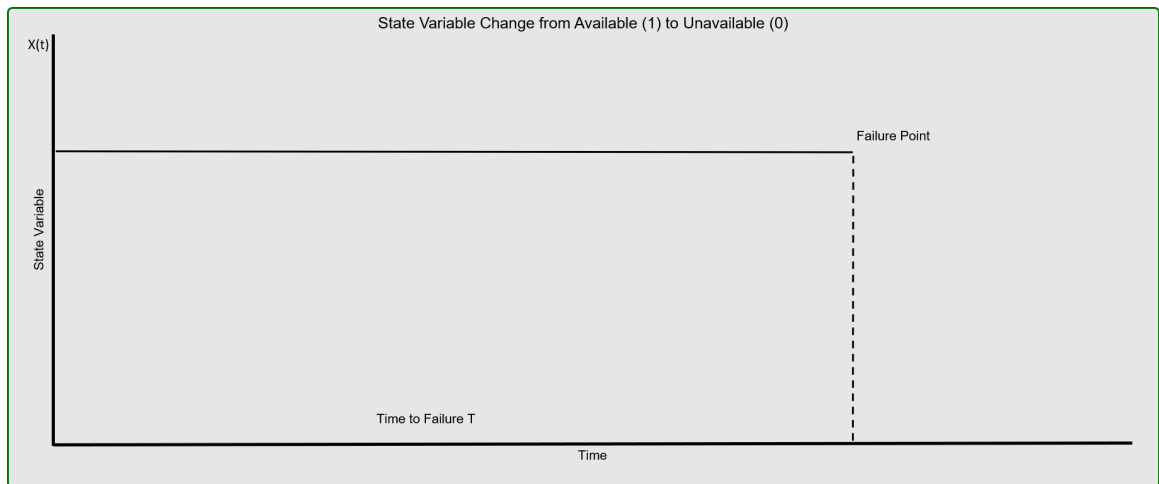


Figure A.2: A Standardised Chart for Time to Failure

standardised set of periods for system state, appears in Figure A.2. Represented graphically, the core periods of system state are;

1. Time 0 with a State of 1 (Item is functioning),
2. Time (T) when the failure occurs (Item is in a failed state) and
3. Time after failure event time.

Within these states, several methods expose qualitative and quantitative failure analysis. System Reliability Theory indicated in Høyland and Rausand (2009) show methods supporting the analysis of failures in a system. Failure Modes, Effects and Criticality (FMEA/FMECA), Fault Tree Analysis, and Cause-and-effect diagrams are a few of the quantitative and qualitative methods used to support system reliability analysis. The anal-

Table A.3: Failure Model Analysis Methods

	Qualitative	Quantitative
FMEA/FMECA	yes	no
Fault tree analysis	yes	yes
Cause and effect diagrams	yes	no
Bayesian belief networks	yes	yes
Event tree analysis	yes	yes
Reliability block diagrams	yes	yes

ysis method used in this thesis approaches systems definition using system decomposition. Thus, fault tree analysis provides suitable pre-conditions and constraint structures within which a binary state. Quantitative decision sensitivity of a system appears in the ISARC. This binary state represents either a functioning or available state (1) or a non-functioning or unavailable state (0).

Failure Analysis Data Sets

Failure analysis and error correlation for distributed systems in research has shown interest in recent years with the research producing a series of data sets located in an online Failure Trace Archive, Epema et al. (2014). These data sets support error correlation analysis across a distributed architecture for several internet-connected system classes, Kondo et al. (2010). At the time of publication of the ISARC model validation, the FTA contained data sets generated from approximately 807000 nodes for a period occurring between 1994 and 2014. A data set summary in Table A.4 expands the summary in Kondo et al. (2010) by applying the trace start dates in ascending order.

In dealing with a distributed systems design, one has to ensure that the correct levels of errors are managed and handled to increase the total reliability of that system. IoT ecosystems are no different in this regard as these systems are primarily distributed. However, the complexity of these systems steadily increases as IoT edge devices may not carry

Table A.4: Summary of FTA data sets adapted from Kondo et al. (2010) and Epema et al. (2014)

System	Type	Nodes Count	Target Component	Period	Year
ucb94	Desktop Grid	85	CPU	46 days	1994
LANL	SMP, HPC Clusters	4750	host	9 years	1996-2005
HPC2	HPC Clusters	256	IO	2.5 years	1996-2005
Microsoft	Desktop	51663	host	35 days	1999
NERSC	HPC Clusters	NA	IO	5 years	2001-2006
Web sites	Web servers	129	host	8 months	2001-2002
Overnet	P2P	3	host	2 weeks	2003
PNNL	HPC Cluster	980	host, network	4 years	2003-2007
Grenouille03	DSL	4800	host	1 year	2003
sdsc03	Desktop Grid	275	CPU	1 month	2003
HPC4	Supercomputers	152516	every thing	1 year	2004-2006
DNS	DNS servers	62201	host	2 weeks	2004
PlanetLab	P2P	200-400	host	1.5 year	2004-2005
Skype	P2P	4000	host	1 month	2005
Grenouille05	DSL	4800	host	1 year	2005
Grid'5000	Grid	1288	host	1.5 years	2005-2006
lri05	Desktop Grid	40	CPU	1 month	2005
deug05	Desktop Grid	40	CPU	1 month	2005
cae06	Grid	686	host	35 days	2006
cs06	Grid	725	host	35 days	2006
glow06	Grid	715	host	33 days	2006
teragrid06	Grid	1001	host	8 months	2006-2007
Docking@home	Desktop Grid	287212	host	8 years	2006-2014
SETI@home	Desktop Grid	226208	CPU	1.5 years	2007-2009
EGEE	Grid	2500 queues	CE queue	1 month	2007
Notre Dame	Desktop Grid	700	CPU, host	6 months	2007

sufficient processing capability or contain the necessary components to recover from internal device failures. Due to this characteristic, these edge devices appear lossy and unreliable. Connection reliability under these conditions varies based on the sensing requirement. Thus, a surge of sensor data from a variable number of connected IoT devices may prove untenable, causing large-scale failures of outbound IoT device transmission requests. Thus, based on the limited nature of process-handling capabilities on constrained IoT devices, a scheme to manage reliable outbound data transmission under failure conditions is critical when stipulating these edge devices' configuration requirements. Table A.4 produces a sample population that creates a point in a time view of reliability metrics and produces configuration options within each system set. For example, the insertion of web browser connection attempts for the 129 websites produced in the FTA. Alternatively, the handling of internal systemic configuration variation in the high-performance computing (HPC) or grid architecture data sets.

A core requirement for a class of managed processes identifies closely with failure analysis. Thus a reference towards state-awareness in components is argued for - similar to those found in Yigitbasi et al. (2010). Yigitbasi et al. (2010) sees the benefits that support the need for a class of state-awareness entities in time-varying behavioural analysis of failures in large scale distributed systems as;

1. having “significant” impact on solutions that handle fault behaviour proactively,
2. having an impact on failure resolution strategies for systems that are in a failed state,
3. being aware of temporal conditions that exist for ‘smart checkpointing’ and system, subsystem, or component level scheduling and
4. having information on creating adaptive fault-tolerant mechanisms.

The design consideration, which outlines the need for information for creating fault-tolerant mechanisms, Yigitbasi et al. (2010), provides the required visibility and, by implication, awareness of the state of a system at any given point in time. The FTA provides a set of failed state occurrences for a DNS, Epema et al. (2014). More specifically, Pang et al. (2004) produces a data sample that references a high number of state switching behaviour in fewer local DNS servers.

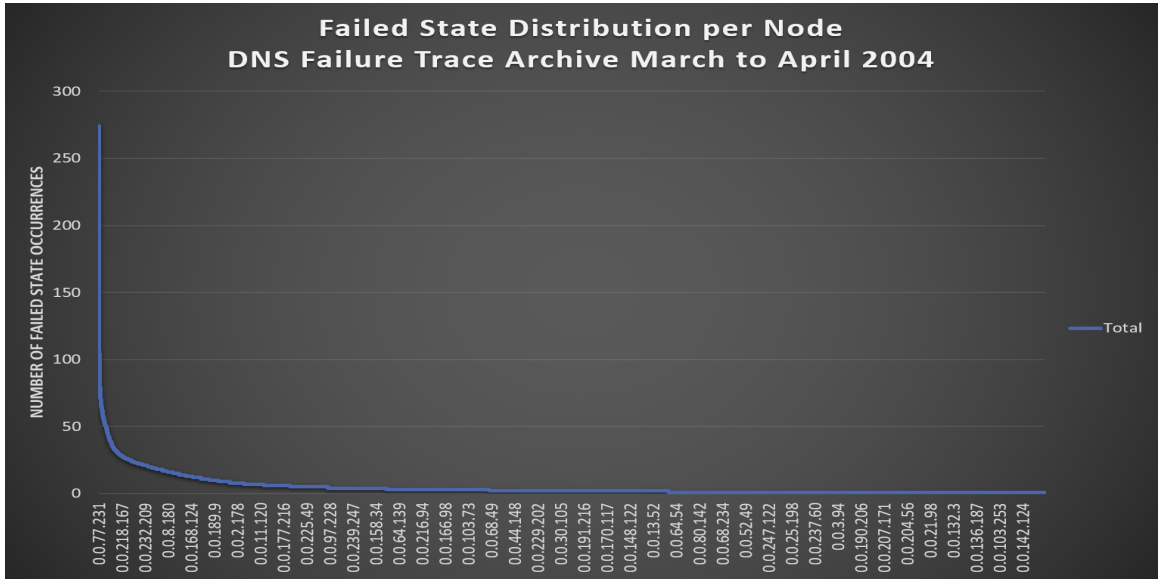


Figure A.3: Failed State Occurrences for DNS Servers. Sampled March to April 2004

Interest in dependability and reliability for IoT systems evolves as a new paradigm due to seamless connectivity requirements, Behera et al. (2015). The reliability of these systems pivots back to core IT system availability metrics as seen in Ahmad (2014). Additionally, Yong-Fei and Li-Qin (2014) presents a quantitative view of IoT reliability by calculating cross-system boundary values for reliability. These boundaries align with current opinions on IoT data generation, transmission, and application domains. The support for greater visibility of the state of components in the IoT edge utilising a purpose-driven approach appears in the ISARC. The design consideration that outlines the need for accessible information to

create fault-tolerant mechanisms seen in the Yigitbasi et al. (2010) time-behavioural analysis, therefore, provides the required state visibility in a hierarchical system template such as the DNS data set in the FTA.

Handling of Omitted Measurement Records in the FTA

In the event of a missing record requirement in the FTA data set, an expansion of the data set into a second-by-second measurement set is possible. The Python code block in listing A.1 below applies a start date and time and end date and available and unavailable states in the DNS data set. An extension to the thesis model proof - its second contribution - enables a series of architectural options representing runtime configurations with sensitivities that attach inside the state boundaries to form a prolonged state. It is however noted that Pang et al. (2004) references the “PASTA” principle. This principle states the ability to observe a given state for the proportion of measurements that are approximations to the amount of time spent in that state. Increases in fixed-rate measurements by Pang indicate that an identical availability distribution is possible in their study. The handling of event insertion provides a measure of generated event surety to create granular measurement opportunities and additional state measurement testbeds. The handling of missing records expands the model’s usefulness by attaching actualised events in a missing data set.

To argue the boundary condition for a coupled parameter set - a runtime configuration - the architectural options condense into an action that occurs well within the start and end state measurements of the FTA data set. To extend the example of logical parameter distances shown in the output *ISARC Parameter List Assignment with Random Generated Sensitivity*, the DNS system communication path or connected IoT device path forms part of a coupled component set. In a connected state, the following sample of suggested parameters could be coupled together; *Address, Communication, Transmission, State:Initial, State:Default, State:Final, Discoverable, Filter, ConnectedDevice, EnrolledDevice*, and so forth. The selection of these parameters forms the implicit knowledge of an IoT device in the connected-edge subsystem based on literature in earlier chapters. One could argue that many more parameters and combinations of these parameters impact the connectivity of IoT devices, and thus manual parameter selection would be untenable. The thesis model proof, therefore, produces an automated sampling of configuration choices. To propose a method of simplifying ongoing architectural decisions, the ISARC adheres to a principle of hierarchy and simplicity. Both principles reflect OPM ISO-19540 standards.

Table A.5 zooms into the original Pang DNS data set showing event_id 6 through 8. Event

7 indicates a start time of the {0:“Unavailability”} state as 1080404578, and the end time as 1080404615. For representation purposes, the code block below uses the event period with the minimum time difference for events in the first ten records of the Pang DNS data set.

A UTC transformation indicates this time as 2004-03-27 16:22:58 and the event stop

Table A.5: Original FTA Data Set Event 6 to 8

Event ID	...	...	Node Name	Event Type	Event Start Time	Event Stop Time	Event End Reason
...	...	...	...	...	...	...	...
6	...	...	0.0.0.1	1	1080376236	1080404578	NaN
7	...	...	0.0.0.1	0	1080404578	1080404615	NaN
8	...	...	0.0.0.1	1	1080404615	1080542754	NaN
...	...	...	...	...	...	...	...

time as 2004-03-27 16:23:35, with a time difference of 37 seconds between these timestamps. The event output table for missing event measurements generates from the code block for

Table A.6: Transformed FTA Data Set Event 6 to 8

EventID	NodeName	EventStart	EventStop	EventType	EventTimeDiff
...	...	...	...	...	...
6	0.0.0.1	2004-03-27 08:30:36	2004-03-27 16:22:58	1	28342
7	0.0.0.1	2004-03-27 16:22:58	2004-03-27 16:23:35	0	37
8	0.0.0.1	2004-03-27 16:23:35	2004-03-29 06:45:54	1	138139
...	...	...	...	...	...

expanding time array records. An additional data set for fitting the second-to-second handling of failure traces appear in the time expansion code block. These missing events enable the testing of architectural choices for each second between {1:“Availability”} and {0:“Unavailability”} state changes, and vice versa.

Listing A.1: Expanding FTA Time Array Records

```

1 #Apply Main FTA data set transformation that yields Intermittent EventDS data set
2 #Input: eventds. Code Block 5.1-DNS Event Trace Transformed Import.
3 #Output: timeArray
4
5 # Expand Missing FTA Data Set Array for Sample event_id 7
6 timeArray = make_array()
7
8 # Event Data Set Event Start Timestamp for Event ID 7
9 stepStart = eventds[3][7]
10
11 # Event Data Set Event End Timestamp for Event ID 7
12 stepEnd = eventds[5][7]
13 createStep = stepEnd - stepStart
14
15 for i in np.arange(createStep+1):
16     timeStep = stepStart+i
17     timeArray = np.append(timeArray, datetime.fromtimestamp(timeStep, timezone.utc).strftime('%Y-%m-%d %H:%M:%S'))
18
19 # Time Array Expansion for event_id 7: 0.0.0.1 2004-03-27 16:22:58 to 2004-03-27 16:23:35
20

```

```

21 #Toggle 3: Print timeArray for missing event timestamps for Event ID 7
22 print(timeArray)
23
24 #['2004-03-27 16:22:58' '2004-03-27 16:22:59' '2004-03-27 16:23:00'
25 # '2004-03-27 16:23:01' '2004-03-27 16:23:02' '2004-03-27 16:23:03'
26 # '2004-03-27 16:23:04' '2004-03-27 16:23:05' '2004-03-27 16:23:06'
27 # '2004-03-27 16:23:07' '2004-03-27 16:23:08' '2004-03-27 16:23:09'
28 # '2004-03-27 16:23:10' '2004-03-27 16:23:11' '2004-03-27 16:23:12'
29 # '2004-03-27 16:23:13' '2004-03-27 16:23:14' '2004-03-27 16:23:15'
30 # '2004-03-27 16:23:16' '2004-03-27 16:23:17' '2004-03-27 16:23:18'
31 # '2004-03-27 16:23:19' '2004-03-27 16:23:20' '2004-03-27 16:23:21'
32 # '2004-03-27 16:23:22' '2004-03-27 16:23:23' '2004-03-27 16:23:24'
33 # '2004-03-27 16:23:25' '2004-03-27 16:23:26' '2004-03-27 16:23:27'
34 # '2004-03-27 16:23:28' '2004-03-27 16:23:29' '2004-03-27 16:23:30'
35 # '2004-03-27 16:23:31' '2004-03-27 16:23:32' '2004-03-27 16:23:33'
36 # '2004-03-27 16:23:34' '2004-03-27 16:23:35']

```

By inserting the missing time events between each event state, the ISARC model proof effectively extends the simulation population without adversely impacting the actual measurements generated in Pang et al. (2004), preserving the failure timestamp positions in the original FTA data set. The ISARC model proof essentially creates a cross-section of the timestamps and expands the measurements such that a boundary between measured states exists. One might argue that an alternative to modelling fixed periods is using a random selection. The thesis model proof argues that random selection between start and end event measurements increases randomness in an already random environment where IoT device placement and IoT device connectedness play a role in connected edge subsystem dynamics. Additionally, the selection criteria for a configuration that contains architectural parameters are sometimes bound directly to an action given a parameter cluster. An example of a parameter cluster is shown in the manual selection of the configuration set containing; *Address*, *Communication*, *Transmission*, $S:\{Initial, Default, Final\}$, *Discoverable*, *Filter*, *ConnectedDevice*, and *EnrolledDevice*.

In Figure A.4, two measurement streams are shown. The Pang DNS data set shows

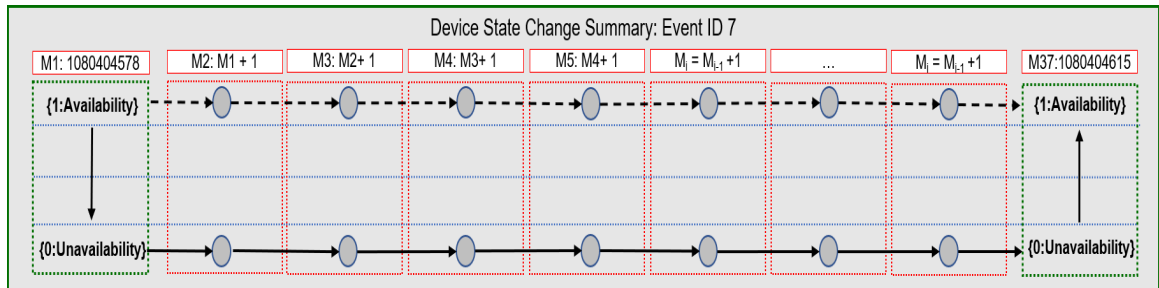


Figure A.4: FTA DNS Event 7 State Change Summary derived from Epema et al. (2014) and Pang et al. (2004)

with relative certainty that a change of state occurred between event_id 6, 7, and 8. In Fig-

Figure A.4 the measurement at M1:1080404578 indicates a state transition from {1:Available} to {0:Unavailable} - from event_id 6 to event_id 7. A subsequent state change shows that M37:1080404615 represents the next state change from {0:Unavailable} to {1:Available} - from event_id 7 to event_id 8. Figure A.4 shows this transition in the solid line transition over the measurement period. The solid line assumes that no process, control, or action occurred during the 37 measurement events to advance the state to anything other than {0:Unavailable}. Thus, the start and end of the measurement period for event_id 7 is accepted as {0:Unavailable}.

The dashed line from M1:1080404578 to M37:1080404615 indicates a single state, but this is not possible in the Pang DNS data set as an {0:Unavailable} state occurred during event_id 7. One could argue that the DNS state changes occurred randomly. However, the Pang DNS data is the basis of DNS reliability calculations that happened at that specific point in time. One could further argue that a randomly generated {0:Unavailable} and {1:Available} data set could provide a comparable basis for the thesis model proof. The thesis model proof is unique. It tests actual events against fixed cluster architectural decisions even though the generation of the thesis model data set uses random principles such as node generation, timestamp generation, and component state at a point in time. Thus, it suggests some determinism in runtime configuration while supporting systems architecture principles such as environmental and systemic OPM characteristics. Environmental system characteristics are largely outside of the control of the core system.

Given the large number of variables that can impact the architecture, a cluster of decisions suggests an understanding of the impact of increasing determinism in the IoT connected-edge device behaviour. Therefore, is there a runtime configuration that includes direct systemic control to change the state earlier than measurement M37:1080404615? Is it cost-effective from a process handling point of view? Is it necessary, or should the IoT device expect a random response when M37:1080404615 produces an {1:Available} response? These questions enable the placement of the controller as systemic instead of environmental. The initial thesis model proposes an environmental placement. By placing the controller within the system boundary, efficiency is achieved as no external communication is required to enable controlled behaviour.

A.2 Node JS Signalling Constants

The NodeJS signalling constants form a critical part of the IoT-Ce interaction decisions. The signalling constants produce an abstracted state machine with defined state transitions

between each of the 36 constants. A system under a control strategy uses the accessible constants to provide the next best action. This action executes a request that solicits a response or an asynchronous notify that initiates a notification to a processor or controlling terminal. The selection of the OS constants occurs under the need for abstracted control towards IoT device. The compatibility of NodeJS on generally accessible IoT devices makes this high-level programming language suitable for accessing device resource information such as device availability, uptime, and battery levels. The direct control of error, failure, and fault transition further appears through remote device management via standards-based application programming interfaces (API). Arguing for structured object and process re-

Table A.7: Consolidated Node JS Signalling Constants

Constant (Interaction Type)	Description
SIGHUP (Notify)	Sent to indicate when a controlling terminal is closed or a parent process exits.
SIGINT (Request)	Sent to indicate when a user wishes to interrupt a process ((Ctrl+C)).
SIGQUIT (Request)	Sent to indicate when a user wishes to terminate a process and perform a core dump.
SIGILL (Notify)	Sent to a process to notify that it has attempted to perform an illegal, malformed, unknown, or privileged instruction.
SIGTRAP (Notify)	Sent to a process when an exception has occurred.
SIGABRT (Request)	Sent to a process to request that it abort.
SIGIOT (Request)	Synonym for SIGABRT
SIGBUS (Notify)	Sent to a process to notify that it has caused a bus error.
SIGFPE (Notify)	Sent to a process to notify that it has performed an illegal arithmetic operation.
SIGKILL (Request)	Sent to a process to terminate it immediately.
SIGUSR1 SIGUSR2 (Request)	Sent to a process to identify user-defined conditions.
SIGSEGV (Notify)	Sent to a process to notify of a segmentation fault.
SIGPIPE (Notify)	Sent to a process when it has attempted to write to a disconnected pipe.
SIGALRM (Notify)	Sent to a process when a system timer elapses.
SIGTERM (Request)	Sent to a process to request termination.
SIGCHLD (Notify)	Sent to a process when a child process terminates.
SIGSTKFLT (Notify)	Sent to a process to indicate a stack fault on a coprocessor.
SIGCONT (Request)	Sent to instruct the operating system to continue a paused process.
SIGSTOP (Request)	Sent to instruct the operating system to halt a process.
SIGTSTP (Request)	Sent to a process to request it to stop.
SIGBREAK (Request)	Sent to indicate when a user wishes to interrupt a process.
SIGTTIN (Notify)	Sent to a process when it reads from the TTY while in the background.
SIGTTOU (Notify)	Sent to a process when it writes to the TTY while in the background.
SIGURG (Notify)	Sent to a process when a socket has urgent data to read.
SIGXCPU (Notify)	Sent to a process when it has exceeded its limit on CPU usage.
SIGXFSZ (Notify)	Sent to a process when it grows a file larger than the maximum allowed.
SIGVTALRM (Notify)	Sent to a process when a virtual timer has elapsed.
SIGPROF (Notify)	Sent to a process when a system timer has elapsed.
SIGWINCH (Notify)	Sent to a process when the controlling terminal has changed its size.
SIGIO (Notify)	Sent to a process when I/O is available.
SIGPOLL (Notify)	Synonym for SIGIO
SIGLOST (Notify)	Sent to a process when a file lock has been lost.
SIGPWR (Notify)	Sent to a process to notify of a power failure.
SIGINFO (Notify)	Synonym for SIGPWR
SIGSYS (Notify)	Sent to a process to notify of a bad argument.
SIGUNUSED (Notify)	Synonym for SIGSYS

quirements within the micro boundary of the component state transition in the Node-JS context, offers a decomposed single boundary opportunity for the interaction between the 36 OS state constants. Applying the OPM ISO-19450, ISO/TC 184/SC 5 (E) (2014), yields the the following object and process suggestions from the records in Table A.7;

1. **SIGHUP** changes state of component from unknown to known when **controlling terminal handling** changes state from open to closed, or when a **parent process exiting** occurs.
2. **SIGINT** changes state of component from running to terminated when **user interrupt requesting** occurs.

Table A.8: IoT Device Operating System Signalling Template

Interaction Type	Constant	1	2	3	4	...
Notify	SIGHUP	1				
Request	SIGINT	2				
Request	SIGQUIT	3				
Notify	SIGILL	4				
Notify	SIGTRAP	5				
Request	SIGABRT	6				
Request	SIGIOT	7				
Notify	SIGBUS	8				
Notify	SIGFPE	9				
Request	SIGKILL	10				
Request	SIGUSR1 SIGUSR2	11				
Notify	SIGSEGV	12				
Notify	SIGPIPE	13				
Notify	SIGALRM	14				
Request	SIGTERM	15				
Notify	SIGCHLD	16				
Notify	SIGSTKFLT	17				
Request	SIGCONT	18				
Request	SIGSTOP	19				
Request	SIGTSTP	20				
Request	SIGBREAK	21				
Notify	SIGTTIN	22				
Notify	SIGTTOU	23				
Notify	SIGURG	24				
Notify	SIGXCPU	25				
Notify	SIGXFSZ	26				
Notify	SIGVTALRM	27				
Notify	SIGPROF	28				
Notify	SIGWINCH	29				
Notify	SIGIO	30				
Notify	SIGPOLL	31				
Notify	SIGLOST	32				
Notify	SIGPWR	33				
Notify	SIGINFO	34				
Notify	SIGSYS	35				
Notify	SIGUNUSED	36				

3. **SIGQUIT** changes state from component from unknown to known when **user termination requesting occurs**, and the sub process **core dump performing** is needed.
4. **SIGILL** changes state of component from unknown to known when **instruction performing** has states; illegal, malformed, unknown, or privileged.
5. **SIGTRAP** changes state of component from unknown to known when **process exception handling** occurs.

Additionally, a DSM equivalent of the state constant interaction extends from the template in Table A.8. The DSM input column (IC) notation reflects an interaction between the row and column values in an n-squared matrix. Thus the row-column intersections of the template in Table A.8 enables sufficient structure to determine interactions, transitions, and state reachability goals in the IoT-Ce.

A.3 A Brief Overview of Systems Control using PID

Appendix A contains PID Control theory considerations in systems design. Appendix A is cited across individual chapters to ensure that the systems architecture and decisions relating to the architecture can decouple cleanly from the system's primary goal, whether IoT data generation, data transmission, or data analytics, Chuah (2014). This decoupling

approach enables a system without binding concepts such as control into architecture decisions. Namely, the thesis does not preemptively argue for systemic versus environmental decision parameters. Control should be one of the many decision parameters that enable sound architectural decisions to support dependability requirements in systems.

A.3.1 Modelling Control Architectures with PID

Feedback is necessary to ensure that systems operate as planned and that intended communication across subsystems and components in a system is reliable. Two types of feedback can exist in a system. One that uses the output of a process to feedback into the system's input to correct over or under adjustment of the control. This type of feedback is generally termed a feedback loop. The second does not use system outputs to reference controller inputs. This loop type is termed a feed-forward loop, as indicated by Janert (2013). Both patterns provide a suitable modelling construct for use cases in IoT. For example, moisture control in agricultural systems where moisture levels in soil can trigger a control action starts to cease sprinkler activity. This interaction is a closed-loop control system. Outbound device state notifications may result from a gateway subsystem needing device status feedback due to a new device in the running IoT ecosystem - in a runtime configuration. Thus, the controller output originating from IoT gateway components within the device could result in a subsequent gateway subsystem control action such as initiating an outbound addressing lookup towards a DNS subsystem. Figure A.5 produces a standard input-output control with reference signal measurements, and corrective control operations.

Figure A.5 outlines a series of specific control system variables. Using a control strategy in the thesis aims to guide the reader to consider that architecture decisions exist to meet a system goal and allow parameter sensitivity to play a role in deciding an optimal architecture. The considerations in the thesis body of work references the use of a novel controller structure to outline adaptive runtime behaviour in the IoT-Ce system use case - enable addressing using a two-tier client/server architecture, with the requisite argument that addressing precedes IoT data transmitting that yields IoT transmitted data. Furthermore, the non-linearity of IoT systems argue for the use of both context and non-context bound control. IoT subsystems and components that have higher processing and memory capacities could hold greater context. In contrast, lossy systems with limited processing and memory capacities would invest in architectures that enable optimised battery and communication component functions.

Figure A.6 outlines the two feedback patterns. *Pattern One*, the **feedback loop**, con-

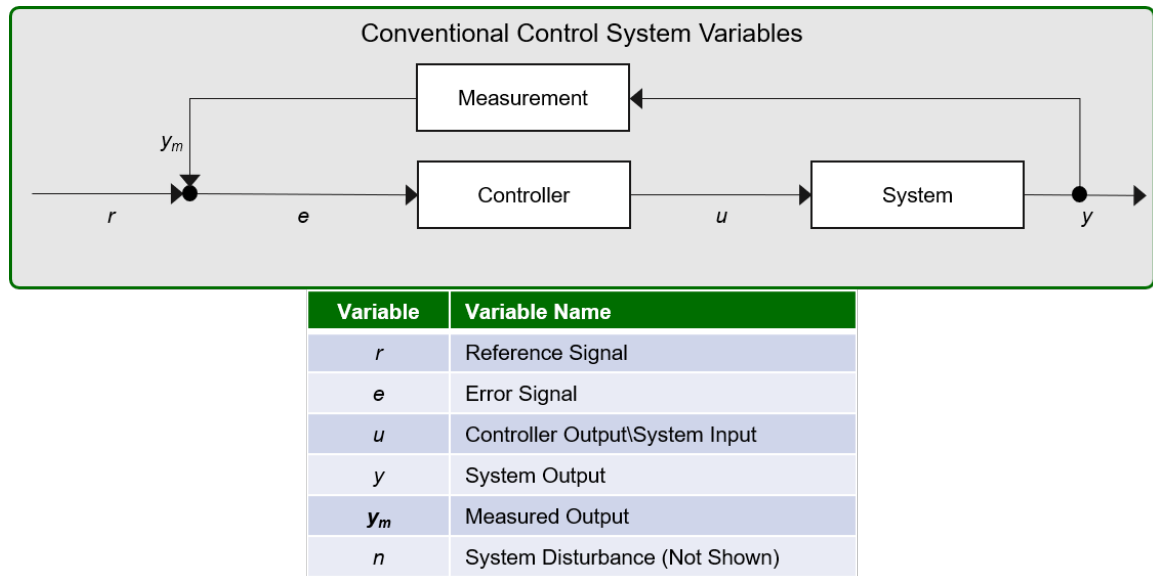


Figure A.5: Conventional Control System Variables. Extracted from Janert (2013)

tains input and mediated inputs through error correction, which serves as returned output-to-input conversions. This corrective measure produces a control target within which a dependability characteristic such as reliability can measure. System behaviour will maintain within a defined error band that defines its reliability.

Pattern Two, the **feed forward loop**, contains no predefined corrected outputs. Reference values can apply, either in or out-of-range of the measured value. However, feed-forward error correction methods can be seen in some control systems through the filtering of signals that may deviate from a required error band.

A control systems viewpoint for electronic computer systems defines the controller's purposes as a model that corrects signalling to be consumed by a system. In his chapter on Controllers, Janert outlines the fundamental definitions for proportional, integral, and derivative control characteristics, Janert (2013)(pp.39-49). These definitions underpin the most widely used model for control systems design - the PID Controller. The characteristics outline as follows;

1. **Proportional Control** outlines the use of a corrective action proportional to the size of the error. Hence, large errors will lead to large corrections, and similarly for small errors. The out of the controller is represented as;

$$u_p(t) = k_p e(t)$$

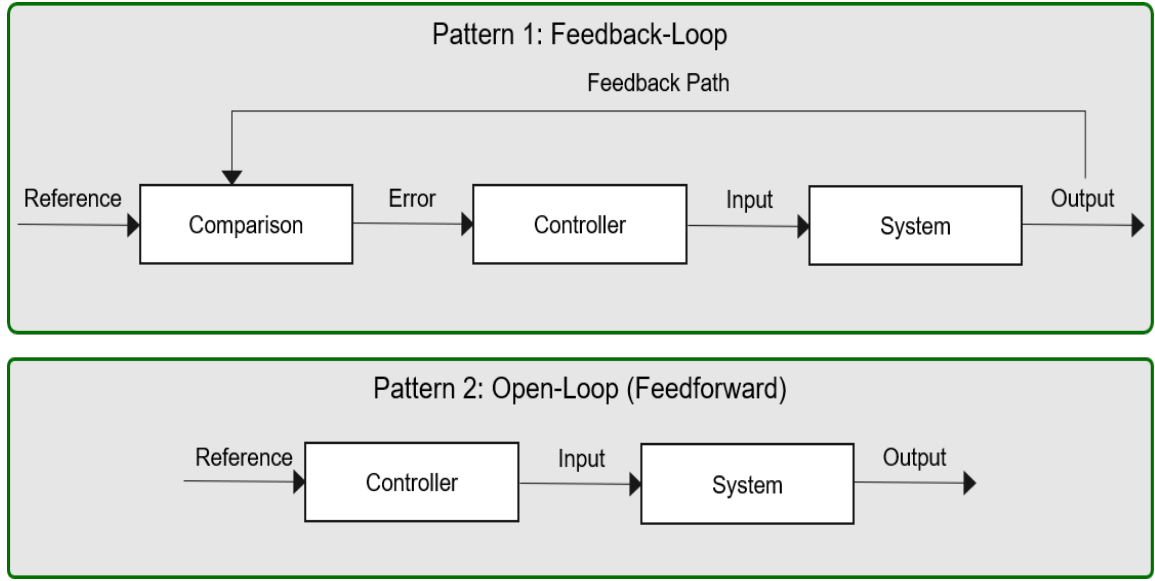


Figure A.6: Closed and Open Loop System Flows. Extracted from Janert (2013)

$$k_p > 0$$

k_p represents the controller gain as a positive constant. The control output $u_p(t)$ can represent a simple yet flawed view of the system's reliability. Although it provides an immediate current state change, it may be useful for improving responsiveness. However, it cannot control an unstable or failed system. Historical context is needed to provide additional control in these cases.

2. **Integral Control** outlines the use of an accumulated error aimed at reducing a steady-state error towards zero.

$$u_i(t) = k_i \int_0^t e(\tau) d(\tau)$$

$$k_i > 0$$

The historical context in an integral control strategy scales small errors so that the error control is significant enough to reduce steady-error states towards zero. The output of the integral is proportional to the error over time. The integral controller is a rudimentary calculation of a cumulative sum of the error values in the system under control. Thus the cumulative error E at time sample t is;

$$E_t = \delta t \cdot e_t + E_{t-1}$$

$$u_{i,t} = k_i E_i$$

Finally, k_i is the integral gain, and $u_{i,t}$ is the implementation of the output of the integral controller at time t .

3. **Derivative Control** outlines the use of a corrective action based on the rate of change of the error - a forward looking concern in the controller model.

$$u_d(t) = k_d \frac{de(t)}{dt}$$

$$k_d > 0$$

Derivative control provides efficient responses compared to integral control actions but cannot accommodate errors where control set points are constants. Derivative control will produce large control signals in response to high-frequency control errors such as set point changes and sensor noise. Thus an early corrective action is needed to stem growing error accumulations. Approximations of a derivative of the error e can apply using;

$$u_{d,t} = k_d \frac{e_t - e_{t-1}}{\delta t}$$

In control systems P, PI or PID controllers are used, however the P and PI is more widely used. The representation for the simulated thesis proof is represented using the standard PID mathematical representation, namely;

$$u_{pid}(t) = k_p e(t) + k_i E_i + k_d \frac{e_t - e_{t-1}}{\delta t}$$

As the thesis control applies a novel PI strategy the result is;

$$u_{pid}(t) = k_p e(t) + k_i E_i + 0$$

Figure A.7 reproduces the system boundary for notification, and PID control parameter usage in OPM form. The notation indicates a separation of concerns between the main system and an externalised controller. As produced in Touretzky (2009), a simulation strategy that produces a reproducible output can be tuned in real-time, in a generally available tool such as Microsoft Excel. PI tuning baselines present values where k_p , k_i , and k_d range from zero to fifty.

Although simulation models can be produced in generally available tools such as Microsoft Excel as seen in Touretzky (2009) and Cameron et al. (2017), the validation of the system architecture model in OPCAT 4.2 is preferred due to this tool's proximity to OPM ISO-19450, Dori et al. (2003). Additionally, OPCAT 4.2 as a single tool for modelling and validation positions the primary system's questions within reach of system architects. The core reason behind implementing the the thesis proof in OPCAT is the concept-to-test principles in existing systems engineering work. This body of work exhibits in the evidence cited in the thesis, such as the use of OPM to determine Cyber-Physical Gaps, Mordecai and Dori (2017a). Finally, the thesis uses Python to enable simulation ready tests for large scale system validation using modular Python Notebooks.

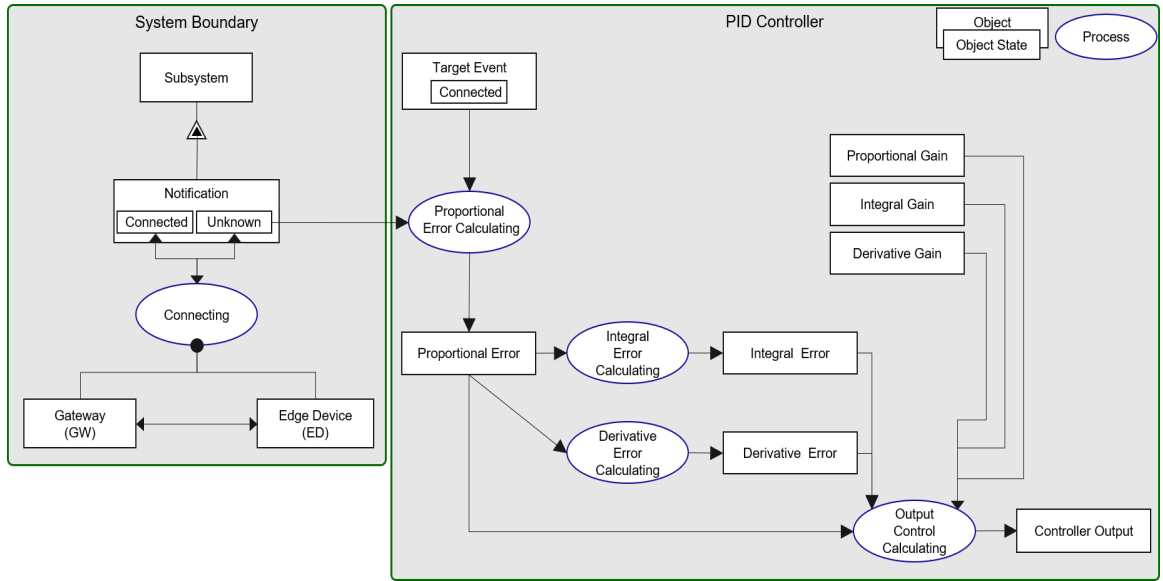


Figure A.7: Simulation Form-Function Definition consolidated from Chapter 4 and 5

A.3.2 Insights on Control Parameters

A model data set can be generated using the control parameters in Table A.9. In this

Table A.9: Control Model Validation Parameters

Parameter Term	Description	DataType
TimeStep	Incrementing Measurement time step	float
ControlPosition	Required control position	float
TargetPosition	Required target position (always at Functioning State 1)	int
Proportional Error (PE)	Control Position - Target Position	float
Integral Error (IE)	Initial Measurement is $(PE * TimeStep + InitialIntegralError)$, then $(PE * TimeStep + PE \text{ for } TimeStep-1)$	float
Derivative Error (DE)	Initial Measurement is $((PE - InitialError) / TimeStep)$, then $((PE - PE \text{ for } TimeStep-1) / TimeStep)$	float
Output	$-((Proportional \text{ Gain} * PE) - (Integral \text{ Gain} * IE) - (Derivative \text{ Gain} * DE))$	float
RandomState	Random State where 1(Functional) or 0(Not Functional)	int
TargetControlValue	$(Output - RandomState) * TimeStep$	float

form, these control parameters serve to meet general controller problems. In an incremental system value to general control design, Stankovic (2014) indicates that human behavioural models can form part of control design for IoT. As part of the control strategy, a control model implementation can reside in several areas within a controlled system; namely, outside of the control loop, inside the controller boundary, inside the system boundary, inside a transducer, and within elected levels in a hierarchical control strategy. However, unplanned internal system variability caused by poor design, inefficient system state-awareness, and incorrect control actions can cause a system to fail in its primary function, irrespective of control placement.

The model parameters in Table A.9 recreates the standard PID control model for motion control studies by deprecating the mass, gravity acceleration, and velocity parameters found in the original control model by Touretzky (2009). This deprecation ensures the necessary PID error calculations still maintain calculation integrity. It further condenses the model to ensure that intermediate acceleration and velocity calculations that impact the actual velocity inputs for the target control state are nullified. Force-model parameters in the baseline PID simulator can eventually be replaced with communication-typed parameters such as network connection delays, processing delays, and round trip time stamps. These parameter variations are possible due to simulation model characteristics in implementing a pure PID controller model.

A.4 State of the Art Control Model Representation

The modelling of control systems in OPM is already represented in Dori’s view on General Systems Theory, Dori (2002)(pp.393). The detailed generic systems architecture proposes a control system foundation using structure and behaviour in OPCAT 4.2. Apart from the OPM-extracted parameter list, the integration of control as either systemic or environmental is a core system boundary argument. In the thesis, the placement of control argues for the existence of control as a supporting function within the OPM System Diagram (SD) hierarchy. The implementation of the concrete control strategy noted in Janert (2013)(pp.161-162), applies an example of scaling capacity to meet the needs of connection requests for an IoT device towards a standard DNS.

The Object Process Language (OPL) outputs the textual leg of a dual-channel form in OPM. Object Process Diagram (OPD) applies the graphical leg containing the primary process, [Affecting](#). The OPD adheres to the modelling principles of singlepage OPD and containing limited intersecting connections.

The use of OPM ISO-19450 applies towards;

1. system architecture **simplification** and **reuse**. OPM ISO-19450 enables a single notation structure and architectural models,
2. systems architecture **usability**. OPM ISO-19450 enables scaling of a complex system down to a single domain that creates a single purpose system boundary with defined interfaces to outofboundary subsystems and components,
3. system architecture **management** and **optimisation**. OPM ISO-19450 enables arguments relating to pre-existing runtime arrangement (runtime configuration) of a system, in order to support non-functional system characteristics,

4. system architecture **management** and **optimisation**. OPM ISO-19450 enables a parameter-based simulation of a decision trade-off model that houses interaction decisions by proposing the integration of OPM and DSM for the IoT-Ce.

Listing A.2: Control System Foundation in OPL. Extracted from Dori (2002)(pp.393)

```

1 Environment is physical.
2 Controlled System exhibits Controlled Attribute.
3     Controlled Attribute is Value.
4     Controlled Attribute exhibits Acceptability.
5         Acceptability can be below limit, within range,
6         or above limit.
7 Controlled System consists of Controller.
8     Controller is physical.
9     Controller consists of Actuator and Measuring Instrument.
10        Actuator is physical.
11        Measuring Instrument is physical.
12        Measuring Instrument can be idle or active.
13    Controller Stores Control Limits.
14 Operator is physical.
15 Operator handles Set Control Limits.
16 Affecting is physical.
17 Affecting requires Actuator.
18 Affecting affects Environment.
19 Affecting changes Acceptability from below limit to above limit.
20 Affecting consumes Decision.
21 Affecting yields Acceptability.
22 Affecting zooms into Decreasing and Increasing.
23     Decreasing yields above limit Acceptability.
24     Increasing consumes below limit Acceptability.
25 Comparing requires Control Limits and Controller.
26 Comparing consumes Measured Value.
27 Comparing yields Decision.
28 Set Control Limits is physical.
29 Set Control Limits yields Control Limits.
30 Measuring consists of Activating, Gauging, and Recording.
31 Measuring requires Controlled System and Clock.
32 Measuring changes Measuring Instrument from idle to active.
33 Measuring yields Measured Value.
34 Measuring zooms into Activating, Gauging, and Recording.
35     Activating changes Measuring Instrument from idle to active.
36     Gauging requires Controlled System.
37     Recording yields Measured Value.

```

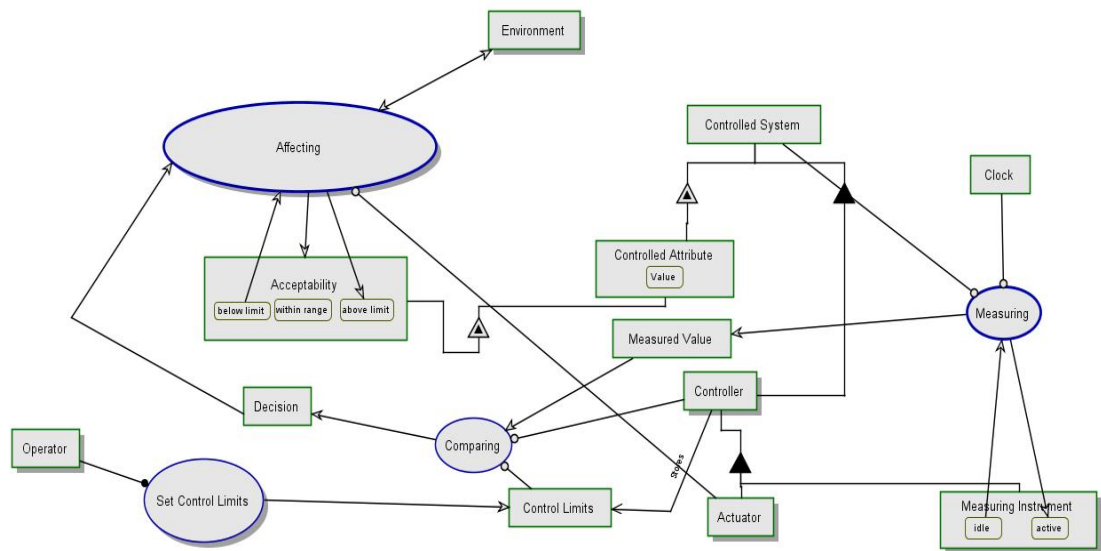


Figure A.8: A generic system under control in OPD. Extracted from Dori (2002)(pp.393)

Appendix B

Additional System Parameters for Runtime Configuration

B.1 ISARC Candidate System Parameters List

Model (re)Production Criteria

The model production parameters outline combinations of symbols in line with partial conformance requirements for the OPM ISO-19450. The main symbols enable;

1. assignment of parameter combinations from Table B.1 using symbolic conformance to objects, processes and states,
2. assignment of parameter combinations from Table B.1 using symbolic conformance to procedural link usage,
3. assignment of parameter combinations from Table B.1 using symbolic conformance to transformation links, consumption links, enabling links, control links and condition links,
4. assignment of parameter combinations from Table B.1 using symbolic conformance to structural links and cardinalities, and
5. assignment of parameter combinations from Table B.1 using symbolic conformance to logical operators.

Table B.1 outlines the main qualitative summary of decision options between environmental and systemic elements in the IoT-Ce. The table maps IoT ecosystem and device requirements from the primary domains in Chapters 1 and 2 into a set of tables in Chapter 3. The consolidated architectural decision parameters appear in Table B.1. In summary, the study extracts known IoT and IoT-Ce requirements from an IoT, Systems Architecture, and Systems Configuration body of knowledge using an OPM-first approach to arrive at a set of

parameters that support ongoing architectural decisions. The parameter-based architecture analysis method by Kazman et al. (1998), supported by Crawley et al. (2016), guides the generation of OPM specific things such as to;

1. present a parameter configuration space for primary IoT-Ce domain scenarios, located in Table B.1,
2. link the parameter space to extract of known subsystem interactions, indicated in Figure 2.1 and Figure 2.11,
3. extract primary function parameter coupling with error limits,
4. extract secondary and supporting function parameter coupling with error limits,
5. model randomness and reliability using first failed decision, namely random event simulation for failed enrolment events,
6. include concrete control parameters, and
7. model sensitivities and architecture trade-off.

The code block in the next section initialises the architectural decision space, which comprises of the parameters that impact ongoing change in the stated 4-model decision architecture. The 4-model decision architecture represents the primary IoT connected-edge system boundary. The integrated parameter set argues for combinations of parameters, essentially, runtime configurations for the IoT connected-edge.

B.1.1 Installing Additional Architecture Parameter Choices

The following variable assignment imports the DSM parameter list from the Chapter 3 definition-To-OPM candidate parameter list into a Python list and then applies a transformation into a Python *NumPy* array. The code block below further shows the inclusion of a random selection sensitivity value for each parameter.

Listing B.1: ISARC OPM Parameter List Assignment

```

1 import numpy as np
2 dsmParam = ['A1.1_Self-configuring', 'A1.2_Communication', 'A1.3_Identifiable Things', 'A1.4_Physical Attribute',
  'A1.5_Virtual Personality', 'A2.1_Connectivity', 'A2.2_Self-configuring', 'A2.3_Heterogeneous Things',
  'A2.4_Identifiable Things', 'A2.5_Physical Attribute', 'A2.6_Virtual Attribute', 'A3.1_Smart Objects', 'A3
  .2_Applications & Services', 'A4.1_Interconnected', 'A4.2_People_Physical Objects', 'A4.3_Pervasive Data
  Collection', 'A4.4_Smart Networking', 'A4.5_Self-configuring', 'B1.1_Acquisition', 'B1.2_Transmission', '
  B1.3_Analytics', 'B2.1_Filter and Collect', 'B2.2_Reader and Scanner', 'B2.3_Sensor', 'B2.4_Actuator', 'B3
  .1_Front End', 'B3.2_Near End', 'C1.1_Communication', 'C1.2_Hardware Abstraction', 'C1.3_Operating System', '
  C1.4_Remote Management', 'C2.1_Data Management and Messaging', 'C2.2_Communication and Connectivity', 'C2.3
  _Application Runtime', 'C2.4_Operating System', 'C2.5_Remote Management', 'D1.1_Notification', 'D1.2
  _Request', 'E1.1_Created', 'E1.2_Running', 'E1.3_Suspended', 'E1.4_Completed', 'E1.5_Activated', 'E1.6
  _Terminated', 'F1.1_Monitor', 'F1.2_Analyse', 'F1.3_Plan', 'F1.4_Execute', 'F1.5_Knowledge', 'F1.5
  _1_Solution Topology Knowledge', 'F1.5_2_Policy Knowledge', 'F1.5_3_Problem Determination Knowledge']

```


Table B.1: Consolidated Full Form-Function Interaction

Source	Parameter	OPM Equivalent Candidate
A1.1	Self-configuring	P:Self-configuring
A1.2	Communication	P:Communicating
A1.3	Identifiable Things	P:Identifying, O:Thing
A1.4	Physical Attribute	P:Attribute Handling, O:Physical Attribute
A1.5	Virtual Personality	P:Personality Handling, O:Virtual Personality
A2.1	Connectivity	P:Connecting or Connection Handling
A2.2	Self-configuring	P:Self-configuring
A2.3	Heterogeneous Things	P:Heterogeneous Thing Handling, O:Heterogeneous Thing
A2.4	Identifiable Things	P:Identifying, O:Identifiable Thing
A2.5	Physical Attribute	P:Attribute Handling, O:Physical Attribute
A2.6	Virtual Attribute	P:Attribute Handling, O:Virtual Attribute
A3.1	Smart Objects	P:Smart Object Handling, O:Smart Object
A3.2	Applications & Services	O:IoT Application, O:IoT Service
A4.1	Interconnected	P:IoT Interconnecting, S:Interconnected
A4.2	People & Physical Objects	O:IoT User Group, O:Physical Object
A4.3	Pervasive Data Collection	P:Pervasive Data Collecting
A4.4	Smart Networking	P:Smart Networking
A4.5	Self-configuring	P:Self-configuring
B1.1	Acquisition	P:IoT Data Generating and Acquiring
B1.2	Transmission	P:IoT Data Transmitting
B1.3	Analytics	P:IoT Data Analysing
B2.1	Filter and Collect	P:IoT Data Filtering, P:IoT Data Collecting
B2.2	Reader and Scanner	P:IoT Data Reading, P:IoT Data Scanning
B2.3	Sensor	P:IoT Data Sensing
B2.4	Actuator	P:IoT Control Actuating
B3.1	Front End	O:Front-end System Set
B3.2	Near End	O:Near-end System Set
C1.1	Communication	P:Communicating
C1.2	Hardware Abstraction	P:Hardware Abstraction Handling
C1.3	Operating System	O:Operating System
C1.4	Remote Management	P:Remote Managing
C2.1	Data Management and Messaging	P:Data Managing, P:Messaging
C2.2	Communication and Connectivity	P:Communicating, P:Connecting
C2.3	Application Runtime	P:Application Runtime Handling
C2.4	Operating System	O:Operating System
C2.5	Remote Management	P:Remote Managing
D1.1	Notification	P:Notifying
D1.2	Request	P:Requesting
E1.1	Created	P:Creating
E1.2	Running	P:Running
E1.3	Suspended	P:Suspending
E1.4	Completed	P:Completing
E1.5	Activated	P:Activating
E1.6	Terminated	P:Terminating
F1.1	Monitor	P:Monitoring
F1.2	Analyse	P:Analysing
F1.3	Plan	P:Planning
F1.4	Execute	P:Executing
F1.5	Knowledge	P:Knowledge Handling, O:Solution Topology Knowledge, O:Policy Knowledge, O:Problem Determination Knowledge

```

3 dsmParameterArray = np.asarray(dsmParam)
4 type(dsmParameterArray)
5
6 ParamProb = np.round(np.random.random(len(dsmParameterArray)),3)
7 ParameterArray = np.append(dsmParameterArray.reshape(len(dsmParameterArray)
8 ,1), ParamProb.reshape(len(dsmParameterArray),1), axis=1)
9
10 ParameterArray
11
12 #Save for Global Variable use
13 np.save('DecisionParameterProperty.npy',ParameterArray)

```

A console printed output of the ParameterArray shows a NumPy ndarray with a shape of (52,2). The second column in the array contains a randomly generated value that exhibits a random sensitivity. The flexibility of using Python to validate the ISARC is that values that exhibit sensitivities closer to a standard or agreed on numeric value can up updated using single or multiple line updates in the array. For example `ParameterArray[42]` results in `array(['E1.5_Activated', '0.02'], dtype=...)`. The modification of a single value is achieved using `ParameterArray[42][1] = np.round(np.random.random(),4)`. A possible output of this modification is `array(['E1.5_Activated', '0.6972'], dtype=...)`. The original 0.02 and the modified

0.6972 are an output of the np.random.random library. However, a standardised sensitivity, probability, or coefficient value for any parameter in the array is possible. As discussed in an earlier section on reliability constructs, reference data appears in references such as Marriott (2002).

Listing B.2: Additional ISARC Parameter List Assignment with Random Generated Decision Sensitivity

```

1 array(['A1.1_Self-configuring', '0.007'],
2       ['A1.2_Communication', '0.497'],
3       ['A1.3_Identifiable Things', '0.704'],
4       ['A1.4_Physical Attribute', '0.756'],
5       ['A1.5_Virtual Personality', '0.52'],
6       ['A2.1_Connectivity', '0.059'],
7       ['A2.2_Self-configuring', '0.359'],
8       ['A2.3_Heterogeneous Things', '0.708'],
9       ['A2.4_Identifiable Things', '0.133'],
10      ['A2.5_Physical Attribute', '0.093'],
11      ['A2.6_Virtual Attribute', '0.044'],
12      ['A3.1_Smart Objects', '0.481'],
13      ['A3.2_Applications & Services', '0.32'],
14      ['A4.1_Interconnected', '0.466'],
15      ['A4.2_People_Physical Objects', '0.597'],
16      ['A4.3_Pervasive Data Collection', '0.71'],
17      ['A4.4_Smart Networking', '0.766'],
18      ['A4.5_Self-configuring', '0.363'],
19      ['B1.1_Acquisition', '0.607'],
20      ['B1.2_Transmission', '0.403'],
21      ['B1.3_Analytics', '0.549'],
22      ['B2.1_Filter and Collect', '0.095'],
23      ['B2.2_Reader and Scanner', '0.196'],
24      ['B2.3_Sensor', '0.874'],
25      ['B2.4_Actuator', '0.045'],
26      ['B3.1_Front End', '0.726'],
27      ['B3.2_Near End', '0.999'],
28      ['C1.1_Communication', '0.587'],
29      ['C1.2_Hardware Abstraction', '0.32'],
30      ['C1.3_Operating System', '0.071'],
31      ['C1.4_Remote Management', '0.985'],
32      ['C2.1_Data Management and Messaging', '0.186'],
33      ['C2.2_Communication and Connectivity', '0.12'],
34      ['C2.3_Application Runtime', '0.915'],
35      ['C2.4_Operating System', '0.286'],
36      ['C2.5_Remote Management', '0.826'],
37      ['D1.1_Notification', '0.324'],
38      ['D1.2_Request', '0.996'],
39      ['E1.1_Created', '0.769'],
40      ['E1.2_Running', '0.177'],
41      ['E1.3_Suspended', '0.227'],
42      ['E1.4_Completed', '0.762'],
43      ['E1.5_Activated', '0.02'],
44      ['E1.6_Terminated', '0.441'],
45      ['F1.1_Monitor', '0.381'],
46      ['F1.2_Analyse', '0.408'],
47      ['F1.3_Plan', '0.526'],
48      ['F1.4_Execute', '0.011'],
49      ['F1.5_Knowledge', '0.663'],
50      ['F1.5.1_Solution Topology Knowledge', '0.204'],
51      ['F1.5.2_Policy Knowledge', '0.831'],
52      ['F1.5.3_Problem Determination Knowledge', '0.622']], dtype='<U38')

```

The creation of the set of parameters found in existing architectures, and structured in the

decision model, is shown in the Python code block above. These architectures are the basis of the parameter extraction process using OPM, which resulted in creating the `dsmParameterArray` array in the code block above. This array attaches to a randomly generated array - `ParamProb` - using a standard NumPy random function. `ParamProb` further uses the number of parameter choices as the length of the array against which the parameters instantiate. Therefore, the number of parameter choices can expand and contract on demand, increasing the reach or depth of the architectural decision space.

The focus of variation of configuration using the additional parameters can suggest a broad scope of combination and composition of these individual parameters. It, therefore, represents a highly non-deterministic state as expected in making architectural decisions. How components interact is also subject to constraints that enforce the interactions within ED/GW components and toward environmental systems such as the DNS. Using logical distances between these selections, sample distribution, and calculating variation of these distances - known statistical constructs - can be implemented using standard distribution data sets. The absolute distance between selections is a construct already outlined in distribution sampling, Adhikari et al. (2020). The support of architectural parameter distancing concurs with the distribution methods cited in the lectures on random sampling and hypothesis testing in Adhikari et al. (2020).

The architectural parameters in the code block above can be generated multiple times, each instance producing a new random value for selection. The distance between the selection of parameters is logical as opposed to experiencing a physical distance. However, this parameter distancing viewpoint is not specifically due to the statistical nature of a random selection of components in the architecture, but rather due to the hierarchical system's form and function and the decisions that impact component selection. As indicated in parameter-based approaches to decisions, the creation of discrete, measurable decision parameters enables a set of qualitative scenarios. These appear through the ATAM, Kazman et al. (1998), and the three decision heuristics in Crawley et al. (2016)(pp.326-327). The thesis simulation method aims to prove the decision strength using parameters, and decisions, that produce the quantitative view of relationships between decision scenarios. This view is supported by creating an architectural decision space, measurable indicators, and segregation of decisions between architecture and design decisions, seen in Crawley et al. (2016). Finally, the scenario interactions limit through the use of DSMs indicated in decision models 1 through 4, which governs a primary system boundary for the IoT-Ce.

In identifying logical distances between parameters from the parameter array, it is noted that IoT device measurements that require 'Communication' '0.xxxx', would therefore also require an 'Address' '0.xxxx'. Thus a positive correlation could exist between decisions on Communication and Address. That is if the boundary condition for the decision is set at 'Arch. Decision 1' '0.500'. If the decision sensitivity leans towards 1, and based on standard probability increasing towards 1, a correlation could be drawn that Communication and Address is positively correlated. A lower sensitivity such as 'Smart Object' '0.xxxx' leaning closer towards 0, can thus be analysed and adjusted based on the need for a decision against the parameter. For example, if the sensitivity of 'Smart Object' is lower than the boundary condition, it can be inferred that the ED/GW is a lossy device with lower processing, memory and storage capacities. Namely not a 'Smart Object', and therefore different class of Communication and Addressing should occur, but Communication and Addressing should occur nonetheless due to the decision correlation between Communication and Addressing. It is argued in the thesis that addressing is a pre-existence condition for communication.

Finally, if Remote Management is required on the IoT device, and a need exists to communicate to establish Remote Management, then the Communication and Remote Management parameters should also be positively correlated. The data set in the code block is truly random, and thus inferences relating to the correlation cannot be made by using these values only. If a 'Communication' '0.905' chance outcome was generated with a 'Remote Management' '0.369' random outcome, it would indicate that 'Communication' '0.905' and 'Remote Management' '0.369' would not correlate as the high, and low values would reduce the chance of these being correlated. Consequently, these would also fall on either side of the 'Arch. Decision 1' '0.500' boundary condition. Finally, the single sensitivity value only shows the selection of a single additional parameter out of the thesis decision array items seen above. A complete decision will focus on decisions relating to other runtime configurations of the IoT device components, which will require coupled components in the subsystem.

A set of trials of the parameters in a single selection form, with random probability, is known to exhibit a normal distribution. Figure B.1 represents 250000 simulations using the code block. Automated simulation is provided as an essential selection methodology that supports continuous model testing, Adhikari et al. (2020). The simulation trails in the code block is adapted from the course work on sampling and hypothesis testing in Adhikari et al. (2020).

Listing B.3: ISARC Sampled Parameter Selection Averages

```

1 import pandas as pd
2 import numpy as np
3 from datetime import datetime, timezone
4 from datascience import *
5
6 selectionCount = []
7
8 ParameterArray = np.load('DecisionParameterProperty.npy')
9
10 isarcParamTmp = Table(make_array('SourceID', 'ParamRef', 'ISARCDecisionLocator', 'ISARC_ParamPValue', '
    ISARC_DetectedPValue'))
11
12 for _ in np.arange(len(ParameterArray)):
13     isarcParamTable = isarcParamTmp.append(['DPA', 'N/A', ParameterArray[_][0], ParameterArray[_][1], 0])
14
15 simulationTrials = 250000
16
17 for i in np.arange(simulationTrials):
18     sampledSelectionAverage = np.average(isarcParamTable.sample(10, with_replacement=False)['ISARC_ParamPValue'
19     ].astype(np.float))
19     selectionCount = np.append(selectionCount, sampledSelectionAverage)
20
21 Table().with_column('Averaged Parameter Selection Value', selectionCount).hist()

```

The graph in Figure B.1 indicates that a sample of ten items (line 19) from a single array containing random selection sensitivity, simulated 250000 time yields a normal distribution. This simple distribution indicates that a single set of static selection criteria array values should yield a normally distributed selection sensitivity.

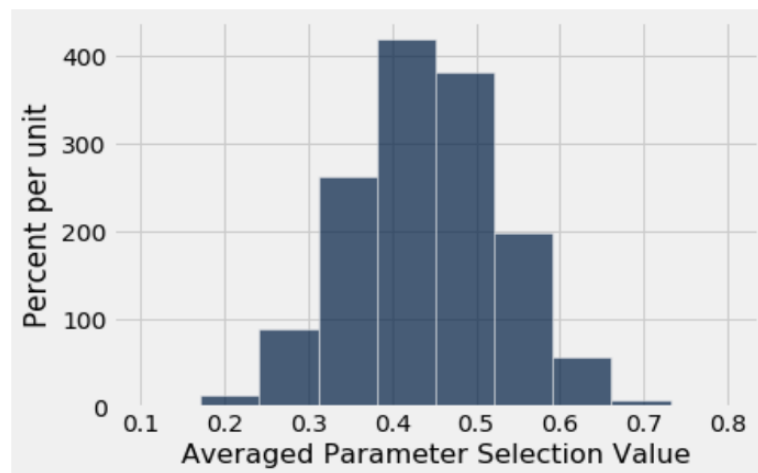


Figure B.1: Sampled Random Parameter Distribution for 52-Parameter Array Table

B.2 Additional Use Case Overview

This section deals with the framework for handling additional enrolment use cases for extending the ISARC OPM-ISO 19450 [Device Discovering](#) functions. In their chapter on interactions, Bassi et al. (2013) produces an IoT architecture for interconnecting things

by discussing two scenario types; *management* and *service-centric scenarios*. Management scenarios focus on the system’s configuration when adding a device to an IoT system and modification of to device configurations. Service-centric scenarios enable the discovery of services within the IoT ecosystem for a connected device. For simplicity, these processes are [System Configuration Handling](#), and [Device Configuration Changing](#), and [Service Discovering](#). The IoT device component decomposes into edge device (ED) and gateway (GW) in the additional use cases, each with its physical system boundary. The ED communicates with a GW using standard connectivity protocols. Thus, a pre-existence criterion for a decision relates to which protocols must exist in the system boundary before communication can occur. Bassi et al. (2013)(pp.239) references these are “Plug & Play” protocols.

B.2.1 Device Discovery - IoT Architecture Reference Model Overview

The Bassi IoT Architecture Reference Model (IoT-ARM) which outlines interacting components in enrolment use cases, is shown in Figure B.2.

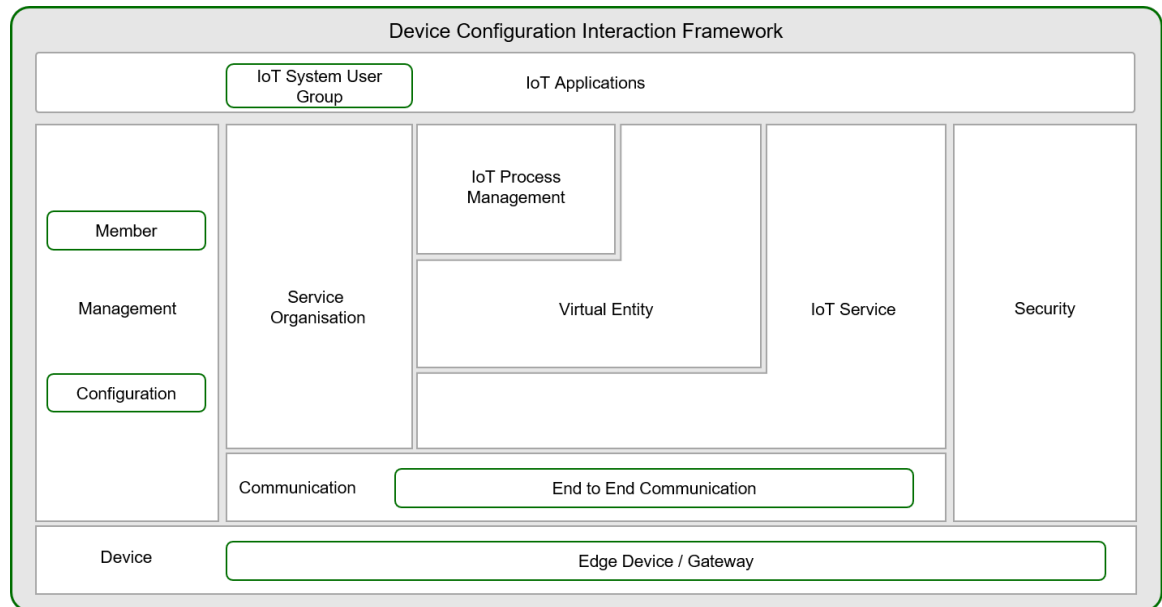


Figure B.2: A simplified layered architecture for IoT system interaction adapted from Bassi et al. (2013)

B.2.1.1 Type 1: Automated Device Discovery and Joining

The automated device discovery pattern outlined in the IoT-ARM, Bassi et al. (2013)(pp. 239), indicates a service discovery method that moves the ED from nil to joined with a minimal number of steps. Shown in Figure B.3, the interaction represents a user moving the ED into the area of a GWs connection range. The EDs internal processes initiate to

advertise ED identification - **Address** - and service arrays. These are the services that the ED houses, such as ambient light, pressure, heat, audio, gas, or other physical quantity sensors. Once advertised, the ED experiences a waiting period within which the GW consumes the service list, performs internal integrity checks, and initiates the enrolment process. This process is necessary to ensure that the GW can schedule service slots to optimise internal GW processing and manage the number of inbound requests from other EDs. These detailed processes unfold within the **Device Range Detecting** process in Figure B.3. Once finalised,

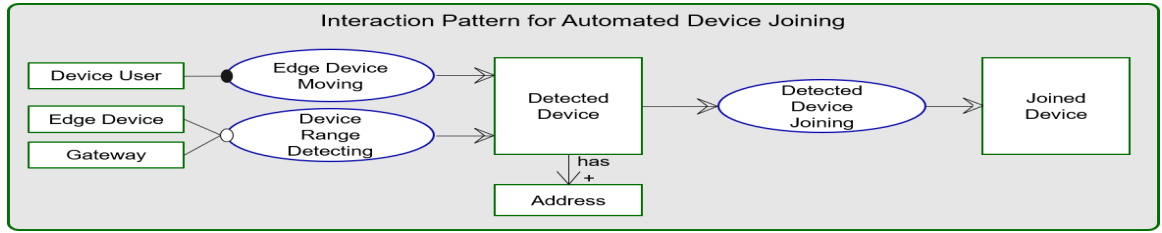


Figure B.3: Interaction Pattern for Automated Device Joining conceptualised from Bassi et al. (2013)(pp. 240)

an addressable discovered device is the input to the **Detected Device Joining** process. Once the ED responds to a connection request, the internal joining process occurs. Finally, a joined ED notification passes to the ED. This notification indicates that the ED is enrolled and ready to send state notifications. This notification could also contain a condensed list of services that the ED can expose based on the earlier service slot scheduling process. By reducing the intended traffic payload earlier, the ED can send lighter responses to ensure more efficient use of its onboard resources. The larger and more robust GW unit can schedule and guide more efficient use of resources. Figure B.3 shows an object-process interaction using a sequence flow. In the flow;

- **Device User** handles **Edge Device Moving**,
- **Detected Device** has at least one **Address**,
- **Edge Device Moving** yields **Detected Device**,
- **Device Range Detecting** requires **Gateway** and **Edge Device**,
- **Device Range Detecting** yields **Detected Device**,
- **Detected Device Joining** consumes **Detected Device**, and
- **Detected Device Joining** yields **Joined Device**.

B.2.1.2 Type 2: Automated Device Discovery and Partially Automated Joining

The partially automated device discovery pattern follows a similar process to a fully automated pattern. However, the joining of a device is queued for an operator to acknowledge service registration on the GW. The automated enrolment pattern executes up to the point before an operator selects the EDs and services that will enroll in the GW. As the GW subsystem components are modular, a partial joining architectural principle creates human and quasi-human processes to acknowledge the registration of services in the GW. Shown in Figure B.4, the queuing of a **Candidate Device** and the required **IoT System Operator** produce the precondition for the joining of an IoT device. The interaction pattern indicates

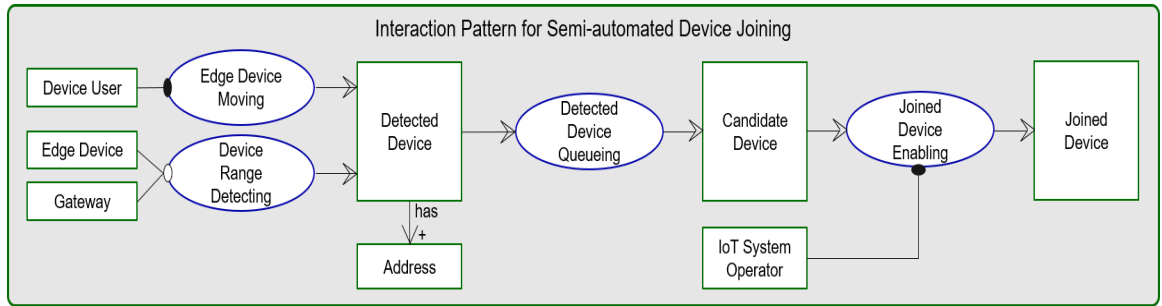


Figure B.4: Interaction Pattern for Semi-automated Device Joining adapted from Bassi et al. (2013)

an extension to the automated joining flow. Bassi et al. (2013) references partial automated joining as adding an addressable device into an allowed firewall list. The addition of allowable addressable devices onto a GW can occur using either human (physical agent) or heuristic (instrumented digital) means. The interaction pattern in B.4 shows that;

- **Device User** handles **Edge Device Moving**,
- **Detected Device** has at least one **Address**,
- **IoT System Operator** handles **Joined Device Enabling**,
- **Edge Device Moving** yields **Detected Device**,
- **Device Range Detecting** requires **Gateway** and **Edge Device**,
- **Device Range Detecting** yields **Detected Device**,
- **Detected Device Queueing** consumes **Detected Device**,
- **Detected Device Queueing** yields **Candidate Device**,
- **Joined Device Enabling** consumes **Candidate Device**, and

- **Joined Device Enabling** yields **Joined Device**.

B.2.1.3 Type 3: Manual Device Joining

Bassi et al. (2013) indicates the manual process for device joining as initiating from a System Operator position. To this end, several IoT platform providers provide a manual device discovery and joining process.

As seen Varga et al. (2018), the Amazon Web Service (AWS) IoT platform houses capabilities for **Device Registering**, **Device Inventory Managing**, and **Policy Enforcing**. Additionally, AWS houses a service called **Device Shadowing** that enables virtual state management of connected-edge devices. This innovation enables devices to appear online with requisite state updates reaching the IoT platform from individual edge devices. The manual device joining primary use case initiates from a Cloud platform towards ED, GW, or ED/GW, hence it does not fall squarely in the ambit of the **Front-end System Set**. Thus the manual joining use case does not scope in the primary thesis arguments. For the sake of completeness concerning the three allowable cases indicated in Bassi et al. (2013), a manual joining interaction pattern is shown in Figure B.5.

The handling of manual device onboarding in a platform ecosystem is seen Varga et al. (2018)(pp.32). Manual data entry is required to capture device information such as name, description, and flexible attribute fields, each holding descriptive form and function detail. Figure B.5 indicates a more substantial final use case for further consideration. Hence, it appears in pure **OPM** (Object Process Method) form. The primary arguments in the claims for using OPM to model the **IoT Systems Architecture for Runtime Configuration (ISARC)** are that using the OPM reduces complexity, enables dual graphical and text output, and enables active learning in systems architecture specification.

Listing B.4: Manual Device Joining in OPL Form

```

1 Edge Device Configuration can be unregistered or registered.
2   unregistered is initial.
3   registered is final.
4 Edge Device Configuration exhibits Device Type.
5 Edge Device Configuration consists of Certificate.
6   Certificate can be not Associated, Associated, detached, attached, or attached Policy.
7     not Associated is initial.
8     attached Policy is final.
9 IoT System Operator handles Device Registration Handling.
10 Device Registration Handling consists of Certificate Associating, IoT Policy Creating, IoT Policy Attaching,
    Device Certificate Attaching, and Device Setup and Rules Configuring.
11 Device Registration Handling requires Certificate and IoT Policy.
12 Device Registration Handling consumes Edge Device Configuration.
13 Device Registration Handling zooms into Certificate Associating, IoT Policy Creating, IoT Policy Attaching,
    Device Certificate Attaching, and Device Setup and Rules Configuring.
14 Certificate Associating changes Certificate from not Associated to Associated.

```

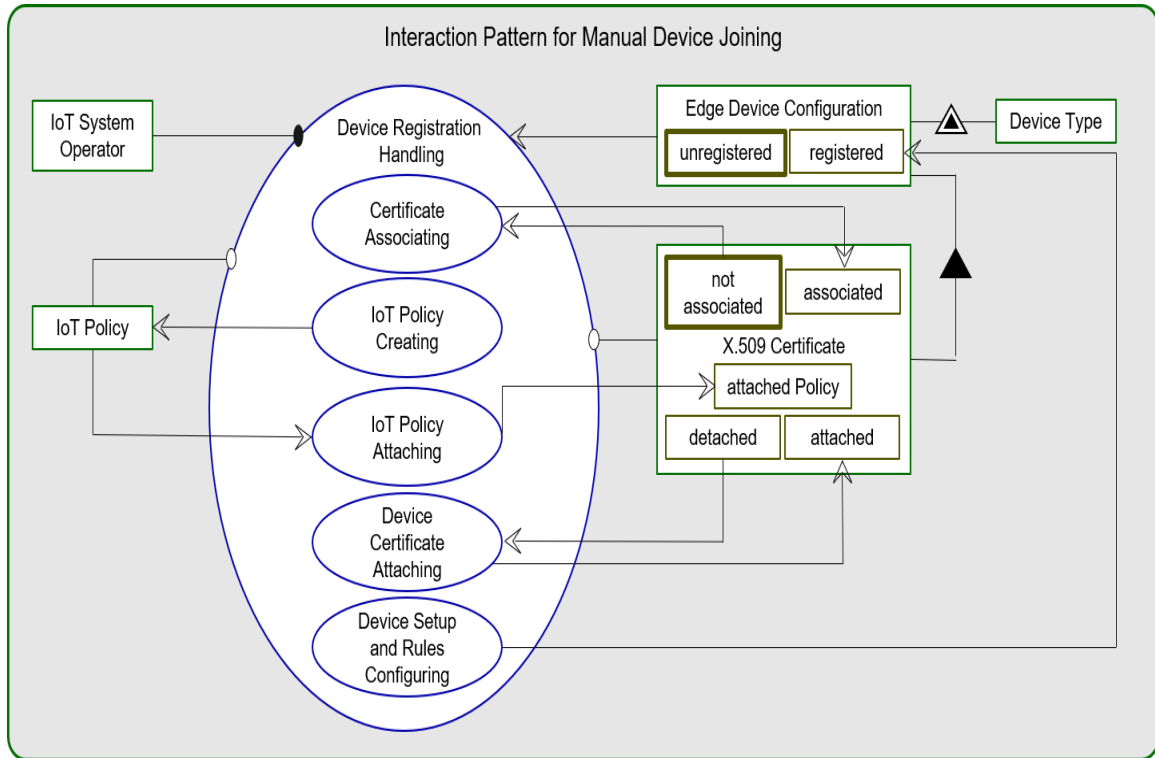


Figure B.5: Interaction Pattern for Manual Device Joining adapted in OPD from Varga et al. (2018)(pp.32)

- 15 IoT Policy Creating yields IoT Policy.
- 16 IoT Policy Attaching consumes IoT Policy.
- 17 IoT Policy Attaching yields attached Policy Certificate.
- 18 Device Certificate Attaching changes Certificate **from** detached to attached.
- 19 Device Setup **and** Rules Configuring yields registered Edge Device Configuration.

References

- Adhikari, A., DeNero, J., and Wagner, D. (2020). Foundations of Data Science: Inferential Thinking by Resampling. <https://www.edx.org/course/foundations-of-data-science-inferential-thinking-b>. [Online; accessed 1-February-2020].
- Adhikari, A., DeNero, J., Wagner, D., and Milner, H. (2018). Computational and Inferential Thinking. <https://www.inferentialthinking.com/chapters/intro>. [Online; accessed 1-February-2020].
- Ahmad, M. (2014). Reliability Models for the Internet of Things: A Paradigm Shift. In *IEEE International Symposium on Software Reliability Engineering Workshops*, pages 52–59.
- Ailisto, H., Arkko, J., Evesti, A., Gurtov, A., Hirvisalo, V., Huusko, J., Jantti, R., Kaasinen, E., Kaipainen, Y., Kantola, R., Kettunen, K., Komu, M., Korhonen, L., Kosonen, P., Kucheryavy, E., Leminen, S., Lilius, J., Numminen, J., Ott, J., and Turunen, M. (2011). Internet of Things Strategic Research Agenda (IoT-SRA). *Finnish Strategic Centre for Science, Technology, and Innovation: For Information and Communications (ICT) Services, businesses, and technologies*.
- Al-Fedaghi, S. and Alahmad, H. (2017). Orientation in Conceptual Modeling Frameworks. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 1298–1303.
- Anderson, J. and Rainie, L. (2014). The 2014 Survey: 2025 Ubiquity - The Internet of Things. A hyperconnected world due to embedded computing, wearables. http://www.elon.edu/e-web/imagining/surveys/2014_survey/2025_Internet_of_Things.xhtml. [Online; accessed 01-Aug-2017].
- Android (2017). Android Architecture. Sensor Stack. <https://source.android.com/devices/sensors/sensor-stack>. [Online; accessed 1-January-2018].
- Anthony, R., Pelc, M., Ward, P., Hawthorne, J., and Pulnah, K. (2008). A Run-Time Configurable Software Architecture for Self-managing Systems. In *International Conference on Autonomic Computing*, pages 207–208.
- Arrow.com (2018). IoT Reference Designs. <https://www.arrow.com/en/reference-designs/>. [Online; accessed 01-Sept-2018].
- Ashraf, Q. M., Habaebi, M. H., Islam, M. R., and Khan, S. (2016). Device Discovery and Configuration Scheme for Internet of Things. In *International Conference on Intelligent Systems Engineering (ICISE)*, pages 38–43.

- Aubrun, C., D, S., and Song, Y. (2010). *Co-design Approaches to Dependable Networked Control Systems*, pages 3,10–11. Wiley-IEEE Press.
- Bahill, A. T. and Madni, A. M. (2016). *Tradeoff Decisions in System Design*, chapter System Design and the Design Process, pages 1–8. Springer.
- Barnaghi, P. and Sheth, A. (2016). On Searching the Internet of Things: Requirements and Challenges. *IEEE Intelligent Systems*, 31(6):71–75.
- Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., and Meissner, S. (2013). *Enabling Things to Talk. Designing IoT solutions with the IoT Architectural Reference Model*. Springer.
- Behera, R. K., Reddy, K. H. K., and Roy, D. S. (2015). Reliability Modelling of Service-oriented Internet of Things. In *4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pages 1–6.
- Bera, S., Misra, S., Roy, S. K., and Obaidat, M. S. (2016). Soft-WSN: Software-Defined WSN Management System for IoT Applications. *IEEE Systems Journal*, PP(99):1–8.
- Bormann, C., Hartke, K., and Shelby, Z. (2014). The Constrained Application Protocol (CoAP). RFC 7252.
- Bradshaw, K. A., Robinson, M., Scazzuso, F., Gallagher, S. M., and Browning, T. (2012). Incorporating Modularity into Ship System Designs for Increased Adaptability. *Maritime Air Systems and Technologies*.
- Brooks, C., Lee, E., Sangiovanni-Vincentelli, A., Sastry, S., C.Tomlin, Bajcsy, R., Hedrick, K., Seshia, S., and Tomizuka, M. (2018). CHESS: Center for Hybrid and Embedded Software Systems. https://ptolemy.berkeley.edu/projects/chess/?_ga=2.264114383.2138253227.1602834406-1016752338.1602834406. [Online; accessed 10-Oct-2020].
- Brooks, C. and Lee, E. A. (2017). CapeCode Tutorial. <https://wiki.eecs.berkeley.edu/accessors/uploads/Main/CapeCodeTutorial.pdf>. [Online; accessed 01-Jan-2017].
- Burgess, M. (1993). Cfengine V2.0 : A network configuration tool. *Theoretical Physics University of Oslo*.
- Burgess, R. and Mende, S. (2005). The Role of Configuration Data and a Configuration Control Module in an End-to-End (E2R) Software Radio System. In *14th IST Mobile and Wireless Communications Summit*, Dresden Germany.
- Cameron, B., Crawley, E., and de Weck, O. (2017). Models in Engineering. <https://sysengonline.mit.edu/course-2/>. [Online; accessed 01-February-2017].
- Cantelon, M., Harter, M., Holawaychuk, T., and Rajlich, N. (2014). *Node.js in Action*, chapter Welcome to Node.js, pages 3–13. Manning Publications.
- Chang, C., Srirama, S., and Buyya, R. (2019). *Wiley Series on Parallel and Distributed Computing*, chapter Internet of Things (IoT) and New Computing Paradigms, page 13. John Wiley and Sons.
- Cheng, S. (2008). *Rainbow: Cost-effective Software Architecture-based Self-adaptation*. PhD thesis, Carnegie Mellon University.

- Chuah, J. W. (2014). The Internet of Things: An Overview and New Perspective in Systems Design. *International Symposium on Integrated Circuits (ISIC)*.
- Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., and Veltri, L. (2014). A Scalable and Self-configuring Architecture for Service Discovery in the Internet of Things. *IEEE Internet of Things Journal*, 1(5):508–521.
- Crawley, E., Cameron, B., and Selva, D. (2016). *System Architecture: Strategy and Product Development for Complex Systems, Global Edition*, chapter 4, 5, 6, 13, 14, pages 63, 68, 98, 126, 310, 326, 327, 335–337. Always learning. Prentice Hall.
- Dart, S. (1991a). Concepts in Configuration Management Systems. In *Proceedings of the 3rd International Workshop on Software Configuration Management*, SCM '91, pages 1–18, New York, NY, USA. ACM.
- Dart, S. (1991b). Overview of Configuration Management for Computer-Aided Design Systems. ftp://ftp.sei.cmu.edu/pub/case-env/config_mgt/slides/cm_for_cad_systems.pdf. [Online; accessed 01-Jan-2018].
- Datta, S. K., Costa, R. P. F. D., and Bonnet, C. (2015). Resource discovery in Internet of Things: Current Trends and Future Standardization Aspects. In *IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 542–547.
- de Matos, H. L. V. (2018). Model-Based Specification of Integrated Modular Avionics Systems using Object-Process Methodology. In *37th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pages 1–8.
- DeNero, J., Culler, D., Wan, A., and Lau, S. (2020). Datascience 0.15.7 documentation. <https://readthedocs.org/projects/datascience/downloads/pdf/master/>. [Online; accessed 24-April-2021].
- Domer, R., Gerstlauer, A., and Muller, W. (2009). Introduction to Hardware-dependent Software Design. In *Asia and South Pacific Design Automation Conference*, pages 290–292.
- Dori, D. (2002). *Object-Process Methodology: A Holistic Systems Paradigm. Systems Theory*, chapter 12, 14. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Dori, D. (2011). Object-Process Methodology for Structure-Behavior Codesign. In Embley, D. W. and Thalheim, B., editors, *Handbook of Conceptual Modeling*, chapter 7, pages 209–258. Springer Berlin Heidelberg.
- Dori, D. (2016a). *Model-Based Systems Engineering with OPM and SysML*, chapter 9, 21, 22, page 77. Springer New York.
- Dori, D. (2016b). Running Model Simulation in OPCAT. http://esml.iem.technion.ac.il/wp-content/uploads/2016/04/Running_simulation.pdf. [Online; accessed 05-June-2019].
- Dori, D. (2019a). Model-based Systems Engineering: Advanced Approaches with OPM. <https://www.edx.org/course/model-based-systems-engineering-advanced-approache>. [Online; accessed 02-May-2019].

- Dori, D. (2019b). Model-based Systems Engineering: Foundations. <https://www.edx.org/course/model-based-systems-engineering-foundations>. [Online; accessed 03-Mar-2019].
- Dori, D., Reinhartz-berger, I., and Sturm, A. (2003). Developing Complex Systems with Object-Process Methodology Using OPCAT. *Lecture Notes in Computer Science*, 2813:570–572.
- Dori, D., Sillitto, H., Griego, R. M., McKinney, D., Arnold, E. P., Godfrey, P., Martin, J., Jackson, S., and Krob, D. (2019). System Definition, System Worldviews, and Systemness Characteristics. *IEEE Systems Journal*, pages 1–11.
- DSMWeb (2019). Design Structure Matrix (DSM). <https://dsmweb.org/design-structure-matrix-dsm/>. [Online; accessed 06-June-2019].
- Eclipse (2016). The Three Software Stacks Required for IoT Architectures. IoT Software Requirements and How to Implement them using Open Source Technology. Technical report, Eclipse.
- Eman, N., McDemid, J., and Bernat, G. (2002). A Technique for Managing Complexity of Use Cases for Large Complex Embedded Systems. In *5th. IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*.
- EPCGlobal Inc. (2008). EPCglobal Object Name Service (ONS) 1.0.1. Technical report, GS1. Ratified.
- Epema, D., Iosup, A., Gallet, M., Jeannot, E., Kondo, D., Javadi, B., and Andrzejak, A. (2014). Failure Trace Archive.For Improving the Reliability of Distributed Systems. <http://fta.scem.uws.edu.au/index.php?n=Main.DataSets/>. [Online; accessed 16-November-2019].
- Eppinger, S. and Browning, T. (2012). *Design Structure Matrix Methods and Applications*, pages 9, 237. MIT Press.
- European Commission CORDIS (2013). Internet of Things Architecture. Final Project Report. http://cordis.europa.eu/project/rcn/95713_en.html. [Online; accessed 01-Aug-2017].
- EvaTech (2019). Teenager Develops Bra to Detect Breast Cancer. <https://www.guidingtech.com/66967/higia-eva-bra-breast-cancer-detection/>. [Online; accessed 09-August-2019].
- Fazio, M., Merlino, G., Bruneo, D., and Puliafito, A. (2013). An Architecture for Runtime Customization of Smart Devices. In *IEEE 12th International Symposium on Network Computing and Applications*, pages 157–164.
- Feiler, P. and Hansson, J. (2007). Impact of Runtime Architectures on Control System Stability.
- Feiler, P. and Li, J. (1998). Consistency in Dynamic Reconfiguration. In *Proceedings. Fourth International Conference on Configurable Distributed Systems (Cat. No.98EX159)*, pages 189–196.
- Gartner (2017). Top 10 IoT Technologies for 2017 and 2018. <https://www.gartner.com/en/newsroom/press-releases/2016-02-23-gartner-identifies-the-top-10-internet-of-things-technologies-for-2017-and-2018>. [Online; accessed 01-Aug-2017].

- Grobshtein, Y. and Dori, D. (2008). 9.2.2 Evaluating Aspects of Systems Modeling Languages by Example: SysML and OPM. *INCOSE International Symposium*, 18.
- GSMA (2015). Humanitarian Connectivity Charter. <https://www.gsma.com/mobilefordevelopment/mobile-for-humanitarian-innovation/humanitarian-connectivity-charter/>. [Online; accessed 25-April-2018].
- GSMarena (2018). Compare Specifications; Samsung Galaxy A8, Samsung Galaxy J7 Prime, Samsung Galaxy Note5. <https://www.gsmarena.com/compare.php3?idPhone1=9432&idPhone2=8314&idPhone3=7431>. [Online; accessed 8-December-2018].
- Haapala, A. (2014). Python Extension for Computing String Edit Distances and Similarities. <https://pypi.org/project/python-Levenshtein/>. [Online; accessed 30-May-2020].
- Henzinger, T. A. and Sifakis, J. (2007). The Discipline of Embedded Systems Design. *IEEE Computer Society*, 1:36–44.
- Horn, G., Eliassen, F., Taherkordi, A., Venticinque, S., Martino, B. D., Bücher, M., and Wood, L. (2016). An Architecture for using Commodity Devices and Smart Phones in Health Systems. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 255–260.
- Høyland, A. and Rausand, M. (2009). *System Reliability Theory: Models and Statistical Methods*, chapter Failure Models, Qualitative System Analysis, pages 16,73–139. Wiley Series in Probability and Statistics. Wiley.
- IBM (2005). MAPE-K. An Architectural Blueprint for Autonomic Computing. Technical report, IBM.
- IBM (2017). IBM and MTN Help Protect Endangered African Rhinos with Internet of Things Technology. <http://www-03.ibm.com/press/us/en/pressrelease/53162.wss>. [Online; accessed 11-Nov-2017].
- IDC (2017). Worldwide Smartphone Market Gains Steam in the First Quarter of 2017. <https://www.idc.com/getdoc.jsp?containerId=prUS42507917>. [Online; accessed 11-Nov-2017].
- IDC (2018). Smartphone OS Market Share. <https://www.idc.com/promo/smartphone-market-share/os>. [Online; accessed 8-December-2018].
- IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std. 610.12-1990*, pages 1–84.
- IEEE (2000). IEEE Standard for In-System Configuration of Programmable Devices. *IEEE Std. 1532-2001 (Revision of IEEE Std. 1532-2000)*, pages 1–130.
- IEEE (2015). Internet of Things (IoT) Ecosystem Study. http://www.sensei-iot.org/PDF/IoT_Ecosystem_Study_2015.pdf. [Online; accessed 25-Aug-2017].
- IncoSE (2018). *A Guide to the Systems Engineering Body of Knowledge (SEBoK) version 1.9.1*.
- Instructables.com (2017). Technology Projects. <http://www.instructables.com/tag/type-id/category-technology>. [Online; accessed 01-Aug-2017].

- IoTOne (2019). Accelerating the Industrial Internet of Things. Use Cases. <https://www.iotone.com/usecases/>. [Online; accessed 9 August 2019].
- ISO/IEC/IEEE (2011). ISO/IEC/IEEE Systems and Software Engineering. Architecture Description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46.
- ISO/TC 184/SC 5 (E) (2014). Automation Systems and Integration — Object-Process Methodology. Standard, International Organization for Standardization, Geneva, CH.
- Jacobson, N. (2004). *The In-system Configuration Handbook: A Designers Guide to ISC*. Kluwer Academic Publishers.
- Jain, K. and Mohapatra, S. (2018). *Edge Computing: From Hype to Reality*, chapter Taxonomy of Edge Computing: Challenges, Opportunities, and Data Reduction Methods, pages 51–69. EAI Springer Innovations in Communication and Computing. Springer International Publishing.
- Janert, P. K. (2013). *Feedback Control for Computer Systems*, chapter Feedback Systems, Scaling Server Instances, pages 15–26, 128, 161–172. O’Reilly Media, Inc.
- Karamanolis, C. T. and Magee, J. N. (1995). Configurable Highly Available Distributed Services. In *14th Symposium on Reliable Distributed Systems*, number 526219, pages 118–127.
- Kaur, N. and Sood, S. K. (2017). An Energy-Efficient Architecture for the Internet of Things (IoT). *IEEE Systems Journal*, 11(2):796–805.
- Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J. (1998). The Architecture Tradeoff Analysis Method. In *4th IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193)*, pages 68–78.
- Kellogg, G. (2019). JSON-ld 1.1. W3C working draft, W3C. <https://www.w3.org/TR/json-ld11/>.
- Khaitan, S. K. and McCalley, J. D. (2015). Design Techniques and Applications of Cyber-physical Systems: A Survey. *IEEE Systems Journal*, 9(2):350–365.
- Khan, W. Z., Xiang, Y., Aalsalem, M. Y., and Arshad, Q. (2013). Mobile Phone Sensing Systems: A Survey. *IEEE Communications Surveys Tutorials*, 15(1):402–427.
- Kharpal, A. (2019). Huawei launches new operating system, says it can ‘immediately’ switch from Google Android if needed. <https://www.cnbc.com/2019/08/09/huawei-launches-its-own-operating-system-hongmengos-or-harmonyos.html>. [Online; accessed 11-August-2019].
- Khazbak, Y., Izz, M., ElBatt, T., Fahim, A., Guirguis, A., and Youssef, M. (2017). Cost-Effective Data Transfer for Mobile Health Care. *IEEE Systems Journal*, 11(4):2663–2674.
- Kondo, D., Javadi, B., Iosup, A., and Epema, D. (2010). The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 398–407.

- Kopetz, H. (2008). The Complexity Challenge in Embedded System Design. *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*.
- Kopetz, H. (2011). *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Springer, Vienna, Austria, 2nd edition.
- Kostic, Z. (2017). DS103x Enabling Technologies for Data Science and Analytics: The Internet of Things. Lesson 2f. Constrained Application Protocol. <https://courses.edx.org/courses/course-v1:ColumbiaX+DS103x+2T2016/courseware/>. [Online; accessed 01-Jan-2017].
- Laplante, P. and Ovaska, S. (2012). *Real-time Systems Design and Analysis. Tools for the Practitioner*. Wiley, 4th edition.
- Latronico, E., Lee, E. A., Lohstroh, M., Shaver, C., Wasicek, A., and Weber, M. (2015). A Vision of Swarmlets. *IEEE Internet Computing, Special Issue on Building Internet of Things Software*, 19(2):20–29.
- Lee, E., Seo, Y., and Kim, Y. (2019). Self-Adaptive Framework Based on MAPE Loop for Internet of Things. *Sensors*, 19(13).
- Lee, E. A., Asare, P., Broman, D., Prinsloo, G., Torngren, M., and Sunder, S. S. (2012). Cyber-Physical Systems - a Concept Map. <https://ptolemy.berkeley.edu/projects/cps/>. [Online; accessed 25-Aug-2017].
- Lee, S., Jeong, J. P., and Park, J. S. (2016). DNSNA: DNS Name Autoconfiguration for Internet of Things Devices. In *18th International Conference on Advanced Communication Technology (ICACT)*, pages 410–416.
- Leon-Garcia, A. (2008). *Probability, Statistics, and Random Processes for Electrical Engineering*. Pearson/Prentice Hall.
- Li, L., Soskin, N. L., Jbara, A., Karpel, M., and Dori, D. (2019). Model-Based Systems Engineering for Aircraft Design With Dynamic Landing Constraints Using Object-Process Methodology. *IEEE Access*, 7:61494–61511.
- Linux (2018). Node.js v10.8.0 Documentation.OS Constants. <https://nodejs.org/api/os.html>. [Online; accessed 01-August-2018].
- Liu, H., Eldarrat, F., Alqahtani, H., Reznik, A., de Foy, X., and Zhang, Y. (2017). Mobile Edge Cloud System: Architectures, Challenges, and Approaches. *Systems Journal*, 1.
- Lohstroh, M. and Lee, E. A. (2015). An Interface Theory for the Internet of Things. In *Software Engineering and Formal Methods*, pages 20–34.
- Lundberg, M. (2018). Welcome to simple-pid’s documentation! <https://simple-pid.readthedocs.io/en/latest/>. [Online; accessed 01-February-2021].
- Maier, M. W. and Rechtin, E. (2009). *The Art of Systems Architecting. 3rd. Edition*. Systems Engineering. CRC Press.
- Marriott, D. L. (2002). 45 - Reliability and Life Testing. In Middleton, W. M. and Valkenburg, M. E. V., editors, *Reference Data for Engineers*, page 1515. Newnes, Woburn, 9th edition.

- Marwedel, P. (2010). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Netherlands.
- Marwedel, P. (2014). Embedded and Cyber-Physical Systems in a Nutshell. In Center, D. K., editor, *47th Design Automation Conference*, pages 1–21. DAC Knowledge Center.
- McLeod, D. (2014). TechCentral. Altech Node: we reveal all the details. <https://www.techcentral.co.za/altech-node-we-reveal-all-the-details/51085/>. [Online; accessed 01-Aug-2017].
- Miorandi, D., Sicari, S., Pellegrini, F. D., and Chlamtac, I. (2012). Internet of things: Vision, Applications and Research Challenges. *Ad Hoc Networks*, 10(7):1497 – 1516.
- Mordecai, Y. and Dori, D. (2014). Model-based Protocol Engineering: Specifying Kerberos with Object-Process Methodology. In *28th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI)*, pages 1–5.
- Mordecai, Y. and Dori, D. (2017a). Minding the Cyber-Physical Gap: Model-Based Analysis and Mitigation of Systemic Perception-Induced Failure. *Sensors*, (17):1644.
- Mordecai, Y. and Dori, D. (2017b). Model-based Requirements Engineering: Architecting for System Requirements with Stakeholders in Mind. In *IEEE International Systems Engineering Symposium (ISSE)*, pages 1–8.
- Oliphant, T. E. (2006). Guide to NumPy. <https://archive.org/details/NumPyBook>. [Online; accessed 1-February-2020].
- Olsson, J. (2014). 6LoWPAN demystified. <http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=swry013&fileType=pdf>. [Online; accessed 01-Aug-2017].
- OMG (2017). OMG Systems Modeling Language. <https://www.omg.org/spec/SysML/1.5/PDF>. [Online; accessed 01-Sept-2017].
- Pang, J., Hendricks, J., Akella, A., Maggs, B., Prisco, R. D., and Seshan, S. (2004). Availability, Usage, and Deployment Characteristics of the Domain Name System. In *Internet Measurement Conference (IMC)*.
- Pei, L., Guinness, R., Chen, R., Liu, J., Kuusniemi, H., Chen, Y., and Kaistinen, L. C. J. (2013). Human Behavior Cognition using Smartphone Sensors. *Sensors*, 13(2):1402–1424.
- Perera, C., Jayaraman, P. P., Zaslavsky, A., Georgakopoulos, D., and Christen, P. (2014). Sensor Discovery and Configuration Framework for the Internet of Things Paradigm. In *IEEE World Forum on Internet of Things (WF-IoT)*, pages 94–99.
- Petrie, C. J. (2012). *Automated Configuration Problem Solving*, chapter Specific Configuration Systems, pages 8, 23–29. SpringerBriefs in Computer Science. Springer New York.
- Plotly Technologies Inc. (2020). Plotly Graphing Libraries. <https://plotly.com/python/>. [Online; accessed 22-May-2020].
- Postscapes (2017). IoT Standards and Protocols. <http://www.postscapes.com/internet-of-things-protocols/>. [Online; accessed 01-Aug-2017].

- Ptolemaeus, C., editor (2014). *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org.
- Qiu, Q., Fleeman, J. A., Ball, D. R., Rackliffe, G., Hou, J., and Cheim, L. (2013). Managing Critical Transmission Infrastructure with Advanced Analytics and Smart Sensors. In *2013 IEEE Power Energy Society General Meeting*, pages 1–6.
- Rammig, F., Ditze, M., P.Janacik, Heimfarth, T., Kerstan, T., Oberthuer, S., and Stahl, K. (2009). *Hardware-dependent Software Principles and Practice*, chapter Basic Concepts of Real Time Operating Systems, pages 15–44. Springer.
- Rao, B., Teja, M., and Nitin, N. (2009). Comparison of Process Scheduling Methodologies for Embedded Systems. *2nd. International Conference on Emerging Trends in Engineering and Technology*, pages 387 – 391.
- Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*, chapter Process-Aware Information Systems, Process Configuration Support, pages 34,89. Springer Berlin Heidelberg.
- Robinson, J. (2004). More Misunderstandings about State Machines [Readers’ response]. *Computing Control Engineering Journal*, 15(5):4–4.
- Rubenstein, D. (2017). DS103x Enabling Technologies for Data Science and Analytics: The Internet of Things. Lesson 3e. Energy Concerns and Harvesting. <https://courses.edx.org/courses/course-v1:ColumbiaX+DS103x+2T2016/courseware/>. [Online; accessed 01-Jan-2017].
- Rumpler, B. and Elmenreich, W. (2006). Considerations on the Complexity of Embedded Real-Time System Design Tasks. *IEEE International Conference on Computational Cybernetics*.
- Rychly, M. (2014). Dynamically Reconfigurable Architectures: An Evaluation of Approaches for Preventing Architectural Violations. In *Handbook of Research on Architectural Trends in Service-Driven Computing*, Advances in Systems Analysis, Software Engineering, and High Performance Computing, chapter 2, pages 26–43. IGI Global.
- Sargent, R. G. (2010). Verification and Validation of Simulation Models. In *Proceedings of the 2010 Winter Simulation Conference*, pages 166–183.
- Schoitsch, E. and Niehaus, J. (2017). CP-SETIS. Strategic Agenda on Standardization for Cyber-physical Systems. <https://artemis-ia.eu/publication/download/cp-setis-standardization-on-cyber-physical-systems.pdf>. [Online; accessed 01-Aug-2017].
- SciPyCommunity (2019). NumPy Reference. <https://docs.scipy.org/doc/numpy/reference/index.html>. [Online; accessed 1-February-2020].
- Seshia, S. and Lee, E. (2012). *Introduction to Embedded Systems. A Cyber-Physical Systems Approach*. UC Berkeley.
- Shaikh, F. K., Zeadally, S., and Exposito, E. (2017). Enabling Technologies for Green Internet of Things. *IEEE Systems Journal*, 11(2):983–994.
- Sharon, A., de Weck, O. L., and Dori, D. (2012). Model-based design structure matrix: Deriving a dsm from an object-process model. *Systems Engineering*.

- Shelby, Z., Arkko, J., Keränen, A., Jennings, C., and Bormann, C. (2016). Media Types for Sensor Measurement Lists (SenML). Internet-Draft Draft-IETF-Core-SenML-04, Internet Engineering Task Force. Work in Progress.
- Sinche, S., Barbosa, R., Nunes, D., Figueira, A., and Silva, J. S. (2017). Wireless Sensors and Mobile Phones for Human Well-being. In *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, pages 1–4.
- Skyttner, L. (1996). *General Systems Theory: An Introduction*. Information Systems Series. Macmillan Education, Limited.
- Smolin, L. (2014). *00:00 Time Reborn*, chapter Doing Physics in a Box, pages 29,39. Penguin Books.
- Sobol, G. and Dori, D. (2020). Improving OPM Conceptual Models by Incorporating Design Structure Matrix. In *2020 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8.
- Soderborg, N. R., Crawley, E. F., and Dori, D. (2003). System Function and Architecture: OPM-Based Definitions and Operational Templates. *Commun. ACM*, 46:67–72.
- Sridhar, S. (2013). Introducing an Enterprise Mobile Reference Architecture. <https://www.slideshare.net/kinvey/mobile-cloud-reference-architecture-for-enterprise>. [Online; accessed 01-April-2018].
- Stankovic, J. A. (2014). Research Directions for the Internet of Things. *IEEE Internet of Things Journal*, 1(1):3–9.
- Subramanian, L. and Katz, R. H. (2000). An Architecture for Building Self-configurable Systems. In *1st Annual Workshop on Mobile and Ad Hoc Networking and Computing. MobiHOC (Cat. No.00EX444)*, pages 63–73.
- Swanson, J., Cohen, M. B., Dwyer, M. B., Garvin, B. J., and Firestone, J. (2014). Beyond the Rainbow: Self-adaptive Failure Avoidance in Configurable Systems. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, pages 377–388, New York, NY, USA. ACM.
- Taleb, I., Dssouli, R., and Serhani, M. (2015). Big Data Pre-Processing: A Quality Framework. *IEEE International Congress on Big Data*.
- TC-SMG, E. (1996). *Digital Cellular Telecommunications System (Phase 2); GSM Network Configuration Management (GSM 12.06)*. European Telecommunications Standards Institute. Version 4.1.1.
- Thiele, L. and Wandeler, E. (2005). Performance Analysis of Distributed Embedded Systems. *Embedded Systems Handbook*, 2.
- Touretzky, D. (2009). PID Controller Simulation and Computation in Excel. <https://www.cs.cmu.edu/afs/cs/academic/class/15883-f17/lectures/cerebellum-controller/pid.xls>. [Online; accessed 15-October-2018].
- Trifa, V., Guinard, D., and Carrera, D. (2015). Web Thing Model. <https://www.w3.org/Submission/wot-model/>. [Online; accessed 08-August-2019].

- Uckelmann, D., Harrison, M., and Michahelles, F. (2011). *Architecting the Internet of Things*, chapter An Architectural Approach Towards the Future Internet of Things. Springer-Verlag Berlin Heidelberg.
- van Zyl, G. (2015). Fin 24 Tech. Failed Altech Node cost Altron millions. <http://www.fin24.com/Tech/Companies/Failed-Altech-Node-cost-Altron-millions-20151007>. [Online; accessed 01-Aug-2017].
- Varga, E., Mijić, D., and D.Drašković (2018). Amazon Web Services (AWS) IoT. In Foster, B., editor, *Scalable Architecture for the Internet of Things*, chapter 2, pages 25–43. O’Reilly.
- Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I., Mazura, M., Harrison, M., and Eisenhauer, M. (2011). Cluster SRA 2011, Chapter Internet of Things Strategic Research Roadmap.
- Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I. S., Mazura, M., Harrison, M., Eisenhauer, M., and Doody, P. (2009). Internet of Things Strategic Research Roadmap.
- Villegas, N., Tamura, G., and Muller, H. (2017). *Managing Trade-offs in Adaptable Software Architectures*, chapter Architecting Software Systems for Runtime Self-Adaptation: Concepts, Models, and Challenges, pages 17–43. Elsevier Science.
- Wagner, F. and Wolstenholme, P. (2004a). Misunderstandings about State Machines. *Computing Control Engineering Journal*, 15(4):40–45.
- Wagner, F. and Wolstenholme, P. (2004b). The authors’ response [More Misunderstandings about State Machines]. *Computing Control Engineering Journal*, 15(5):5–5.
- Washizaki, H. and Tei, K. (2019). IoT System Architecture:Design and Evaluation. <https://www.edx.org/course/iot-system-architecture-design-and-evaluation-2>. [Online; accessed 28-June-2019].
- Yigitbasi, N., Gallet, M., Kondo, D., Iosup, A., and Epema, D. (2010). Analysis and Modeling of Time-correlated Failures in Large-scale Distributed Systems. In *11th IEEE/ACM International Conference on Grid Computing*, pages 65–72.
- Yong-Fei, L. and Li-Qin, T. (2014). Comprehensive Evaluation Method of Reliability of Internet of Things. In *9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 262–266.
- Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., and Yang, X. (2017). A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, PP(99):1–1.
- Zhang, Y., Qiu, M., Tsai, C., Hassan, M. M., and Alamri, A. (2017). Health-CPS: Healthcare Cyber-Physical System Assisted by Cloud and Big Data. *IEEE Systems Journal*, 11(1):88–95.
- Zheng, X., Julien, C., Kim, M., and Khurshid, S. (2017). Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems. *IEEE Systems Journal*, 11(4):2614–2627.