

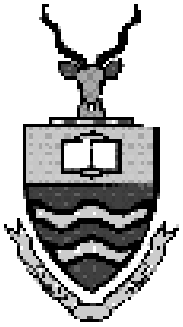
https://dspace.org/

The design and implementation of a Tina Based Asp service for the Satina Trial

Ip, Chris Chung Hang

2006-03-02
http://hdl.handle.net/10539/214

Downloaded from a DSpace repository. https://dspace.org/



UNIVERSITY
OF THE
WITWATERSRAND,
JOHANNESBURG

The Design and Implementation of A TINA Based ASP Service for the SATINA Trial

Chris Chung Hang Ip

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, February 2001

Declaration

I declare that this project report is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Signed this ___ day of _____ 20___

_____

Chris Chung Hang Ip.

Abstract

As the Application Service Provider (ASP) becomes increasingly popular in the software market, this project proposes an ASP service, Remote Software Management Service (RSMS), which is based on the distributed computing environment. The uniqueness of this service is that it uses the Telecommunication Information Networking Architecture (TINA) concepts for the specification, design, implementation and deployment of the service. The method used for developing the RSMS is the Description Plane Model (DPM), which contains five planes namely the Objective Plane, the Definition Plane, the Design Plane, the Implementation Plane and the Physical Plane. The output of the planes is a system consisting of a number of service providers, each having its own roles and functions in the system. The software of the system is implemented and deployed on the South Africa TINA trial (SATINA) platform. By applying the TINA concepts on the service design and implementation, most of the problems of the traditional ASP service providers can be resolved.

Acknowledgements

I want to thank Prof. H. Hanrahan for giving me the opportunity to study this course. In addition, without my parent's moral and financial support I would not be able to finish this report. I also want to my colleagues for building the SATINA platform together with me. Specifically Jimmy for providing the Object Oriented database for subscriber and service management; Russell, Brandon, Kersten and John for building some part of the retailer reference points and DPE; Fillipe for providing me the sample JAVA code for the client components; Shaun for converting the client JAVA application into JAVA applet; and Justin for providing the documentations on the JAVA codes. Thanks to staff of the Language Lab of the English department of the university for proof reading of my report.

Many thanks to Telkom, Siemens and THRIP who supported me in studying this course. I also want to thank the people in Easysoft, especially Claire, for sponsoring me the full version of the ODBC Bridge Software, which is one of the important components I employed to implement my project design.

Contents

Declaration i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures ix

List of Tables xi

Acronyms xii

1 Introduction 1

1.1 The need for ASPs 1

1.2 Different types of ASPs 2

1.3 Current ASP Business Model 2

1.4 Shortcomings of the current ASP model 3

1.5 Problem Statement 4

1.6 TINA Overview 5

1.6.1 TINA System Framework 5

1.6.2 General TINA Business Model 6

1.6.3 TINA Session Concept 7

1.6.4 Benefits of using TINA for designing the RSMS 8

1.7 Implementation Methodology 8

1.8 Report Overview 9

2 Objective Plane **11**

2.1 Introduction 11

2.2 SATINA ASP Business Model 11

2.3 Stakeholder, roles and obligations 13

2.4 Policies 13

2.5 SATINA RSMS Business Relationship instance 15

2.6 Use Case Diagrams 17

2.6.1 Service Usage 18

2.6.2 Service Maintenance 19

2.6.3 Third Party Sessions 20

3 Definition Plane **22**

3.1 Introduction 22

3.2 The Information Model 22

3.2.1 The User Partition 24

3.2.2 The Content Provider Partition 25

3.2.3 The Integrated Retailer Service Provider Partition 27

4 Design Plane **31**

4.1 Introduction 31

4.2 Computation Model 31

4.3 Service Logic 34

4.3.1 Pre-Service Events 35

4.3.2 Main Service Events 36

4.3.3 Auxiliary Service Events 45

5 Implementation Plane **48**

5.1 Introduction 48

5.2 Engineering model 48

5.3 Implementation Code 51

5.3.1 Basic Object Types 52

5.3.2 IA, UA, PA and SF Implementation 54

5.3.3 USM Implementation 54

5.3.4 SSM Implementation 56

5.3.5 CompUSM Implementation 59

5.3.6 Retailer PA Implementation 60

5.3.7 RSMSCore Implementation 62

5.3.8 ssUAP Implementation 65

5.3.9 ODBC Bridges and Database Engine Implementation 68

5.3.10 Retailer and 3rd party Admin GUI Implementation 70

5.3.11 Web Interface Implementation 74

6 Physical Plane **76**

6.1 Introduction 76

6.2 Technology Model 76

6.2.1 Hardware Technology 76

6.2.2 Software Technology 77

6.3 Physical Deployment Model 81

6.4 Topology 83

6.5 Applicable Application Software 85

7 Conclusion **87**

7.1 Discussion 87

7.2 Conclusion 88

7.3 Future Work 89

References **91**

A Sequence Diagrams **92**

B IDL definitions **96**

B.1 RSMSCommonTypes 96

B.2 TINA3rdpartyCommonTypes 97

B.3 TINA3rdptyInitial 99

B.4 TINAProvider3rdpartyUsage 101

B.5 RSMSRetUSM 103

B.6 RSMSRetSSM 105

B.7 RSMSCUSM 107

B.8 RSMS3rdUSM 109

B.9 RSMS3rdSSM 111

B.10 RSMSCore 113

C Tested Applications **116**

Bibliography **117**

List of Figures

1.1 Current ASP Business Model 3

1.2 TINA DPE and Underlying Node 5

1.3 TINA Business Model 7

1.4 Service Design Planes 9

2.1 RSMS Business Model 12

2.2 Business Relationship Instance Diagram 15

2.3 Service Usage Use Case Model Package 18

2.4 Service Maintenance Use Case Model Package 19

2.5 Third Party Sessions Use Case Model Package 20

3.1 Information Model 23

3.2 Information Model - User Partition Highlight 24

3.3 Information Model - Content Provider Partition Highlight 26

3.4 Information Model - Integrated Retailer Partition Highlight 27

4.1 Computational Model 33

4.2 Main Service Flow 36

4.3 Sessions and Domains 38

4.4 Request Software List 40

4.5 Remote Execution Flowchart 41

4.6 Modify Software Entry 46

4.7 Usage Record Query by User 47

5.1 DPE Architecture 49

5.2 Node, Capsules and Clusters 50

5.3 Computational Object, eCO template, and eCOs 50

5.4 Deploying Computational Object to Engineering Object 51

5.5 Login Dialog 66

5.6 Software List Dialog 67

5.7 Software Details Dialog 67

5.8 Relational Database Tables 69

5.9 Ret Admin GUI Main Menu 71

5.10 Add Application Dialog 72

5.11 Application List Dialog 73

5.12 Usage Records List Dialog 74

5.13 Web Interface for Usage Record Queries 75

6.1 Topology 83

6.2 Logical Network Planes 84

A.1 Remote Execution 93

A.2 Start Third Party Sessions 94

A.3 Destroy Third Party Sessions 95

List of Tables

2.1 Stakeholders, Roles and Obligations 14

5.1 Additional Interfaces and Client for RSMS USM 55

5.2 Additional Interfaces Required by RSMS USM 55

5.3 Additional Interfaces and Client for RSMS SSM 57

5.4 Additional Interfaces Required by RSMS SSM 57

5.5 Interfaces and Client for compUSM 59

5.6 compUSM Required Interfaces 60

5.7 Additional Interfaces and Client for Retailer PA 61

5.8 Additional Interfaces Required by Retailer PA 61

5.9 Interfaces and Client for RSMSCore 62

5.10 RSMSCore Required Interfaces 62

5.11 Interfaces and Client for RSMS ssUAP 65

5.12 RSMS ssUAP Required Interfaces 65

6.1 Components Deployment Table 82

Acronyms

ASP	Application Software Provider
compUSM	Composer User Session Manager
CORBA	Common Object Request Broker Architecture
DPM	Description Plane Model
GUI	Graphical User Interface
IA	Initial Agent
IDL	Interface Definition Language
ISP	Internet Service Provider
ISV	Independent Software Vendor
MAP	Managed Application Provider
MFC	Microsoft Foundation Classes
ODBC	Open Database Connectivity
OMG	Object Management Group
ORB	Object Request Broker
PA	Provider Agent
POA	Portable Object Adapter
QoS	Quality of Service
RDP	Remote Desktop Protocol
RM-ODP	Reference Model for Open Distributed Processing
RSMS	Remote Software Management Service
SATINA	South African TINA Trial
SF	Service Factory
SQL	Structured Query Language
SSC	Special Service Component
SSM	Service Session Manager
ssUAP	Service Session User Application
TINA	Telecommunication Information Networking Architecture
UA	User Agent
UML	Unified Modeling Language
USM	User Session Manager

Chapter 1

Introduction

One of the purposes of the next generation networks is to support different kinds of services, in addition to the traditional voice service. An example of such services is the Application Software Provider (ASP) service, which is becoming more popular in the present market [5]. However there are some limitations of the current ASP implementations due to the existing network architecture and traditional business model. In order to resolve these issues, the background of the ASP must be considered.

1.1 The need for ASPs

The concept of ASP is as old as computers themselves. In the early days when the cost of hardware was unaffordable for most companies, mainframes were designed to host a particular application where relatively cheap, dumb terminals were connected to the mainframe sharing its expensive processing time. Nowadays, companies are looking for alternative ways to share their expansive resources –the software, as the cost of ownership of software becomes higher and higher.

Besides sharing of software resources, the main drive behind the ASP market is the tremendous growth of the Internet, which creates the basic distributed computing platform for the ASP services. Although the Internet brings the connectivity from the client to the server, it is far from ideal. This kind of connectivity introduces concerns such as insecurity and unreliable quality of service.

The convergence of the telecommunication networks and the Internet brings the concept of next generation networks, such as the TINA platform, which may be used to improve the service offered by the present ASPs. At the same time, the

convergence also opens up the doorway for different parties, such as traditional Telcos, to fight fiercely against each other for the ASP market.

1.2 Different types of ASPs

The Internet and the ASP industry have been growing so fast that almost everybody with different fields of expertise are rushing into the business. This results in different "types" of ASPs such as the ISP/Telcos, ASP pure play or independent software vendors (ISVs). Since most of the ASPs only specialise in their own field, they normally lack the necessary experience in other fields. For example the ASP pure play may be very strong in developing and hosting applications but may not have a sound understanding of customer support such as ISP/Telcos do. In contrast, ISP/Telcos may have a solid communication infrastructure but may not be able to develop as flexible an application as ISVs would.

Normally a single ASP may not be able to offer everything that a particular firm requires and multiple ASPs are needed. Thus, integrating and communicating between applications from different ASPs are gaining in importance. In addition to the integration problems, when a company decides to switch from one ASP to another, porting the hosted applications and their data to the new ASP should also be seriously considered.

1.3 Current ASP Business Model

There are two main ASP models: one offers the traditional standalone software packages through the use of the client-server architecture such as Citrix client and Microsoft Terminal Server, and X-client and X-server. The other model is the web-based application such as the web based e-mail services. In the first case the applications are running in the server side and the outputs (such as screen and sound) are "exported" to the client terminal using a proprietary communication protocol[1], as if the applications are running locally on the client side. The second case allows the potentially millions of terminals equipped with a standard browser to interact with the application server remotely.

The current models, as shown in figure 1.1, usually involve either two or three parties, namely the client, the ISP and the ASP, or the client and a combined ISP/ASP

[1]Windows systems use Remote Desktop Protocol(RDP) while Unix systems use X-Protocol

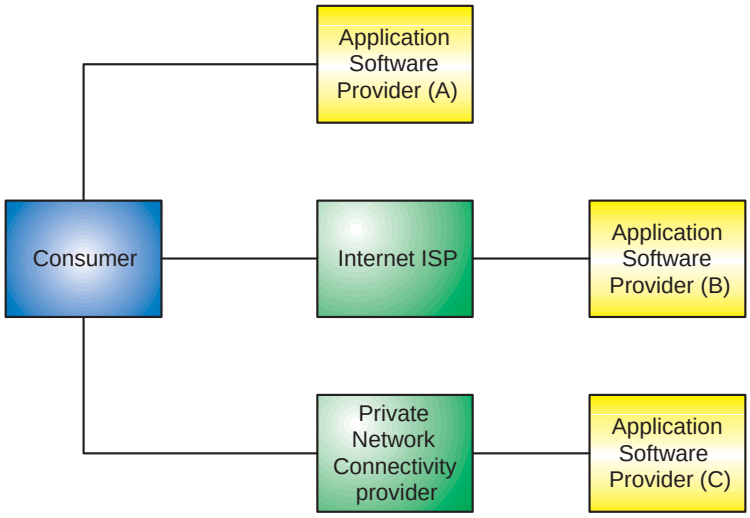


Figure 1.1: Current ASP Business Model

provider. A single client can be served by multiple ASPs which are normally running in parallel and independently. As most ASPs are competing against each other, they usually operate their supporting functions, such as accounting, billing, network management, database administration, in their own proprietary way. The client company might even have to install different network connections and gateways in order to communicate with different providers.

1.4 Shortcomings of the current ASP model

The shortcomings of the current ASP model can be summarised as follows:

Quality of service - reliability problems associated with the Internet Protocol and the public portion of the network result in mission critical applications not suitable for hosting.

Interoperability between ASPs - it is possible that ASPs do not have a transport medium or might not have adequate bandwidth between each other. Different authentication mechanisms implemented by different ASPs makes it difficult to communicate between each other. Lack of common interfaces or reference points also complicates communication.

Compatibility between ASPs - the method of accomplishing the same task is not compatible among different ASPs . For example, data format stored in different ASPs might not be compatible between one another. This leads to

difficult data exchange between ASPs and the reluctance of the client to switch from an unsatisfactory ASP to another competing ASP.

Integration with third parties - due to lack of interfaces and weak definitions of business relationships between different parties, the current ASP model prevents the integration of different parties from creating a pure electronic e-commence environment. For example, without integration, an ASP hosted application cannot charge the customer for the goods through the bank automatically. It cannot issue an electronic delivery order to the courier service company to deliver the goods. If integration is needed, it is presently done using proprietary interfaces and communication protocols between both sides of the system. As a result of the non standard interfaces, the ASP is bound to a specific third party and cannot switch to other players without a major modification of the existing system.

Weakness in certain fields - since the ASP may involve many different disciplines, such as software development, traffic analysis, customer training and support etc, a single ASP may not have adequate experience or manpower to serve all these areas efficiently and effectively.

Security - inherent insecurity of the Internet causes sensitive data to be vulnerable to unauthorised access.

1.5 Problem Statement

This project specifies, designs, implements and deploys an experimental ASP service called the Remote Software Management Service (RSMS) to address the limitations of the current ASP implementations, by adopting the concepts and principles from the Telecommunication Information Networking Architecture (TINA). The implementation is to be deployed on the South African TINA Trial (SATINA) platform.

As TINA covers many different aspects of a distributed computing environment, it is very difficult to follow and implement all the concepts involved. Thus, only the TINA service architecture and computational architecture are applied in the development of the RSMS. Other TINA defined architectures, such as the network resource architecture and management architecture, are considered in the service design but they are not fully implemented.

1.6 TINA Overview

The TINA architecture provides a set of concepts and principles that can be applied in the specification, design, implementation, deployment, execution, and operation of software for the telecommunication systems in the distributed computing environment. The objectives of TINA include the provision of a software architecture that offers reusable software components; supports network-wide software interoperability; reduces the lead time to introduce new services to the market; and hides the heterogeneity of the underlying technologies from the service designer. Due to the distributive nature of the ASP service, the TINA architecture is the ideal candidate to be applied to designing such a service.

The TINA architecture is very broad in its scope and in-depth in detail. In this introduction, only those concepts that the RSMS benefits most are discussed. More detailed guidelines for TINA concepts can be found in [3] and [7].

1.6.1 TINA System Framework

The TINA based system is made up of several layers. These layers provide different levels of abstraction to describe the components of which the system comprises. Figure 1.2 shows the layers of a TINA system.

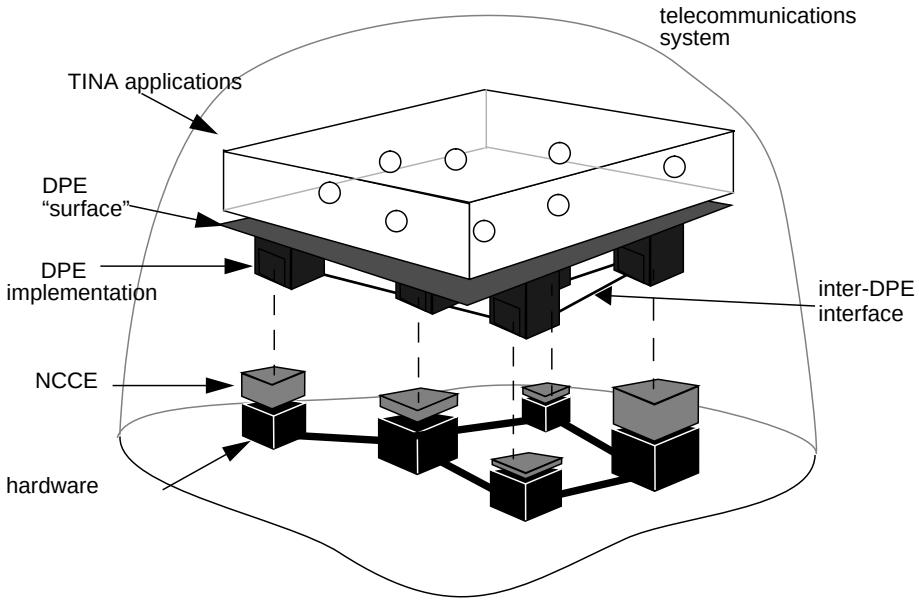


Figure 1.2: TINA DPE and Underlying Node

The topmost layer is the application layer where the TINA application resides. A

TINA application is defined as a set of interacting objects, which is located in the application layer.

The second layer is the Distributed Processing Environment (DPE), which is the software that supports the distributed execution of the objects that comprise of the TINA applications. The DPE provides uniform communication mechanism and basic services to the objects.

The DPE is actually created by a number of computing nodes that execute the DPE software. The third layer, namely the DPE implementation layer, consists of these nodes. The Kernel Transport Network (KTN) is used to provide the communications between these nodes.

Under the DPE implementation layer is the Native Computing and Communications Environment (NCCE) which describes the computing environment, such as the operating system, within a particular node. This layer is used to support the execution of DPE and non-DPE software and to interface the DPE with the underlying communication devices.

The lowest layer is the layer where the hardware resources exist. These hardware resources are responsible for the physical exchange of information between the computing nodes. Examples of the hardware resources are the network adapters and switches. Special hardware, such as the real time video encoder, is also included in this layer.

1.6.2 General TINA Business Model

The TINA business model is a general framework for use to identify the appropriate roles, reference points and related interfaces for any TINA service. It is based on the RM-ODP enterprise viewpoint, which specifies the roles of the actors of the system. It provides the guideline for defining the scopes and boundaries of different parties involved in a service. The model also defines some consistent interfaces between different parties. These interfaces facilitate interoperability between different systems. Five main roles are defined in the TINA business model, namely the consumer, retailer, broker, connectivity provider and third party service provider, as shown in figure 1.3.

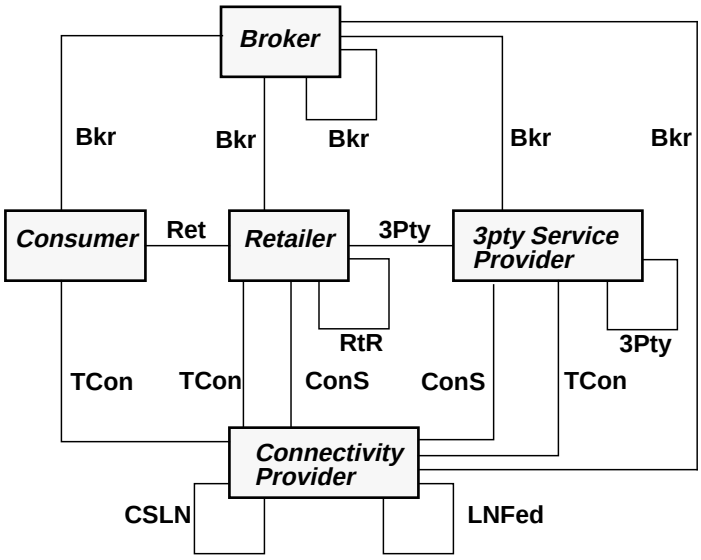


Figure 1.3: TINA Business Model

1.6.3 TINA Session Concept

Even though the services are different, they provide a fundamental property that is a context for relating activities. Such a context is regarded as a *Session*. The generic definition of term "session" stated in [3] is:

> A temporal period during which activities are carried out with the purpose of achieving a goal.

Four types of sessions have been identified by TINA: *Service Session*, *User Session*, *Communications Session*, and *Access Session*. The Service Session represents a single activation of a service. The User Session represents a single user's interaction with a Service Session. The Communications Session represents the connections associated with a Service Session. The Access Sessions represents a user's attachment to a system and his involvement in its services.

The purposes of these session concepts are to separate out different concerns and to promote distribution of functionality. For example, separating the Access Session and Service Session allows the access methods and technology for different users to vary. It also provides generic access methods to different Service Sessions. The separation of the Service Session and the User Session permits the distribution of functions and state. The storing of state permits the suspension and resumption of the service. The separation of the Service Session and the Communication Session supports the division of the service functions from the physical connections involved.

1.6.4 Benefits of using TINA for designing the RSMS

The benefit of using the TINA business model is that it provides standard reference points for the different parties involved in a service, such as the data repository provider, the confidence provider and managed application provider (MAP), and integrating them as a single TINA application to provide new services. Different ASPs can communicate between each other using the retailer to retailer (RtR) reference points. In addition, they can share the service offered by third party service providers. For example, they can share the same data repository provider in order to access the same database to maintain data integrity. The relationships can be dynamically established implying that no one service is bounded to a specific provider.

The business model also facilitates the cascaded outsourcing solution. For example, the ASPs might outsource other MAPs to manage the set of applications they host. In this way the specialist companies can offer services with their fields of expertises while developing other components to compete with other service providers. For instance, existing ISP/Telcos can implement the standard interfaces as the connectivity provider while exploring the ASP market by developing the retailer interfaces for the ASP services separately.

Using the TINA service architecture, different domains (e.g. ASP and MAP) have to gain access to other domains using the same access session mechanism. The same confidence provider can be used to authenticate the identity of different parties. Thus, trusted relationships between different parties can be established. The session concepts ease the integration between different parties as they promote distribution of functionalities.

The architectural separation of TINA allows the quality of service and other networking issues such as location transparency and transportation protocol to be isolated from the service design. Therefore, service providers, such as the RSMS, do not have to consider these issues when designing the service. The problems are left to the underlying network connectivity provider to handle.

1.7 Implementation Methodology

This project uses the Description Plane Model (DPM)[3] as the design guideline for the RSMS. The DPM defines five planes of development where each plane is used

to create certain development activities. The output of the planes of the DPM is the abstract language constructs that can be used to express the design of a system. The output models of the DPM are based on the five viewpoints[2] of the Reference Model for Open Distributed Processing (RM-ODP). These output models are the essential building blocks of a TINA service.

Alongside with DPM the elements of Unified Modeling Language (UML), such as the Use Case Diagrams, State Diagrams and Sequence Diagrams, are integrated in the planes. The additions of the UML elements strengthen the concepts and ideas behind the development activities. These elements fit very well into different planes of the DPM and they support object oriented programming and the implementation of the RSMS service.

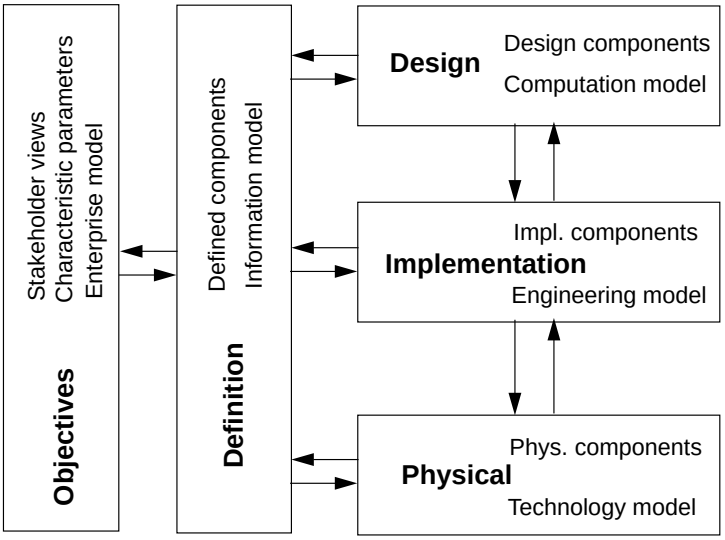


Figure 1.4: Service Design Planes

1.8 Report Overview

The main body of this report is divided into five chapters, corresponding to the five planes of the DPM. The first plane of the development cycle, which is discussed in chapter 2, is called the objective plane. This plane is concerned with the roles that people and organizations will be carrying out with respect to the service and its supporting environment. Within this plane, the business model is defined. In addition, the Use Case Diagrams of the UML are used to further describe the roles of each participant of the RSMS service.

[2]The five viewpoints are: Enterprise, Information, Technology, Engineering and Computation

Chapter 3 relates to the second plane of DPM named the definition plane, which constructs an object-oriented description of the service and its environment. The information model is defined in this plane.

The third plane is the design plane, which contains object-oriented descriptions of the components of logic and data that will be used to implement the service. This plane also contains the computational model. The sequence diagrams and state diagrams defined in UML are used to illustrate the interactions of the computational objects. This plane is discussed in chapter 4

Chapter 5 describes the implementation plane, which defines the deployable software modules comprising of the service on a distributed platform. The outputs of the implementation plane are the engineering model and the implementation code. Screen outputs are also described in this chapter.

Chapter 6 deals with the last plane of the DPM, which is the physical plane. This plane concerns the operational deployment and execution of the software modules. The physical plane maps to the technology viewpoint of the RM-ODP standard, which requires detailed system information such as programming language, the hardware platform, operating system and network protocols. This chapter describes the technology employed on the SATINA platform and how the RSMS components are deployed on the platform.

Although the structure of this report shows the logical sequence of the development process, the development of the design plane, implementation plane and physical plane are in parallel order as can be seen in figure 1.4. The reason for the parallel development of the planes is because the information needed by these planes is interdependent. For example, one of the outputs of the implementation plane is the implementation code, which cannot be completed without the knowledge of the required programming language that is defined in the physical plane. Therefore, there are overlapping areas which may not seem logical. However, cross-references of these areas are provided to ease the reading of this document.

Chapter 2

Service Design: Objective Plane

2.1 Introduction

The first plane of the development cycle specified by the DPM is called the objective plane, which is concerned with the roles that people and organizations will be carrying out with respect to the service and its support environment. Within this plane, the business model is defined. This plane corresponds to the enterprise viewpoint of the RM-ODP model.

2.2 SATINA ASP Business Model

The RSMS implementation tries to follow the TINA business model as closely as possible. Figure 2.1 shows an example business model that can reflect the typical business scenario. It is based on the general TINA business model discussed in section 1.6.2. It shows the business relationship of different service providers that are integrated together to provide a single RSMS service. It shows stronger bonds between the service providers than the current ASP business model shown in figure 1.1.

In figure 2.1, six different parties are identified as the key parties involved in constructing the RSMS.

The first party identified is the user, who represents the consumer in the business model. The consumer is the party that requires the use of a particular service.

The second and third participants are the RSMS Service Provider and the ASP

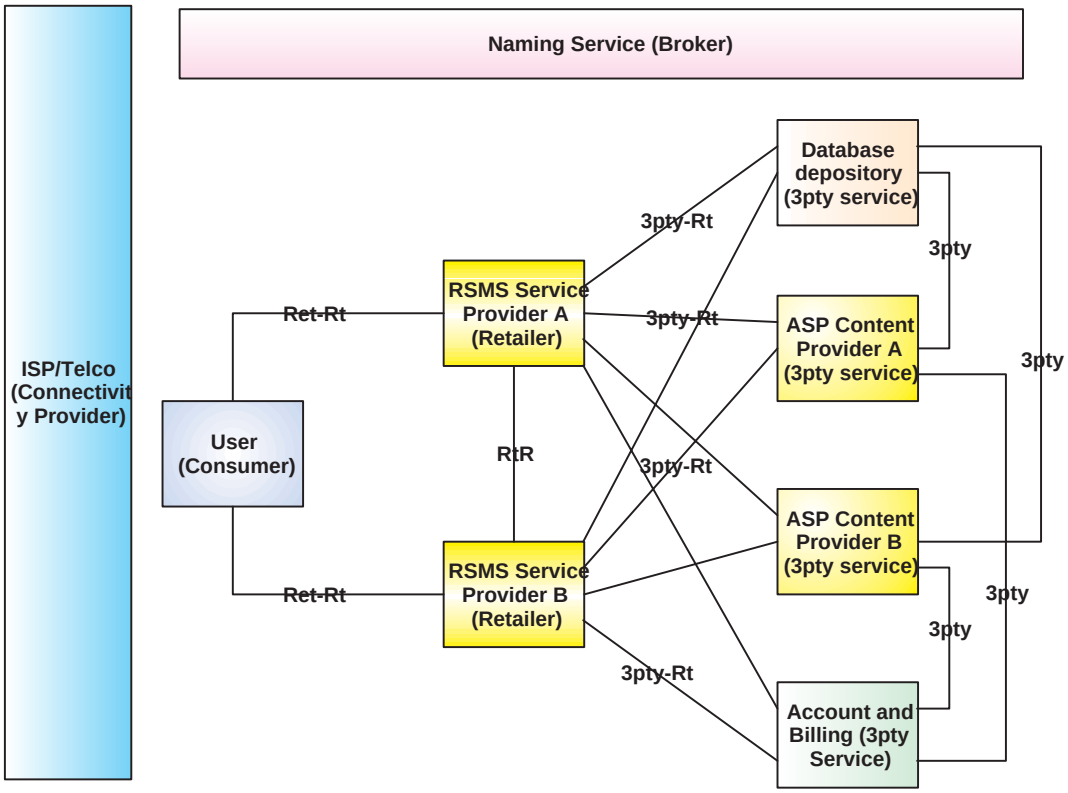


Figure 2.1: RSMS Business Model

Content Provider respectively. The RSMS Service Provider acts as the retailer while the ASP Content Provider acts as the third party service provider. The main function of RSMS Service Provider is oriented towards customer management and value adding. On the other hand, the ASP Content Provider presents and maintains the service content, which is to execute the hosted applications and export the visual outputs to the user terminal.

The fourth party participates in the system is the naming service, which is a cut down version of the broker function in the TINA business model discussed in section 1.6.2. The function of the naming service is to enable the ASP retailer to locate the available ASP third parties within the system.

The fifth party is the database provider, which is acting as a third party service provider. It provides database functionalities to the other parties who need to access the database.

The last party identified is the connectivity provider, which provides the physical connection to all the parties involved. To reduce the complexity of figure 2.1, the lines joining the connectivity provider to the other parties are not shown.

2.3 Stakeholder, roles and obligations

The *stakeholder, roles and obligations* summary is one of the outputs of the objective plane. It clearly defines the responsibility of different stakeholders participated in delivering the RSMS service. This discrete separation of responsibility is important for an environment that requires cooperation and federation of service providers. Table 2.1 lists the stakeholders that participate in the RSMS service and their roles and obligations.

2.4 Policies

Policies are the rules that all the parties should follow while providing or utilising the service. The followings are the basic policies that should be observed by relevant parties.

1. The service provider has the right to determine which user has access to the service of the content providers.
2. The content provider has the right, over the retailer, to determine who can access the contents it provides, in this case the application software.
3. The content provider has the right to add, modify and remove the application software entries in the central database, only if that entry belongs to that content provider.
4. The content provider has the right to review all the usage records in the central database only if the records are related to that content provider.
5. The service provider has the right to remove any application software entries in the central database, no matter which third party content providers they belong to.
6. The service provider has the right to modify any application software entries, provided that a notice is given to the concerned third party content provider. The concerned third party has the right to remove that modified entry from the central database afterward.
7. The content provider cannot terminate the executing application software without informing the user. Enough time should be given to the users to save their current work.

Table 2.1: Stakeholders, Roles and Obligations

Stakeholder	Roles	Obligations
User	Consumer	- Subscribe the service - Check if usage record is correct - Pay for the service to retailer - Choose correct application software to use
Service Provider	Retailer	- Query the naming service for available third party providers - Subscribe to service offered by third party providers - Subscribe to the database provider to create a central database for the service - Manage the content of the central database - Control client access - Pay the third party providers service charges
Content Provider	Third Party	- Register its presents on the naming service - Update its available application software on the central database - Inform the retailer about its availability status - Provider streaming of remote software control protocol between the client and its application server - Control retailer access - Ensure the usage is recorded correctly on the central database
Database Provider	Third Party	- Maintain the database service such that down-time is minimum - Restrict access to the database
Naming Service	Broker	- Provide retailer fair access of the presents of third party content providers - Restrict access to the service
Network Operator	Connectivity Provider	- Provide kernel transport connection between different parties - Provide streaming connections between the user and the content provider - Maintain the service up-time and Qos

8. The user has the right to check and verify his own usage records.

9. The usage records cannot be modified unless agreed by all three parties namely the user, the content provider and the service provider.

2.5 SATINA RSMS Business Relationship instance

After all the policies and obligations are defined, a specific business relationship instance of the RSMS service is produced. This business relationship is specific to a particular instance, in this case the SATINA implementation, which defines the reference points relationships between different stakeholders.

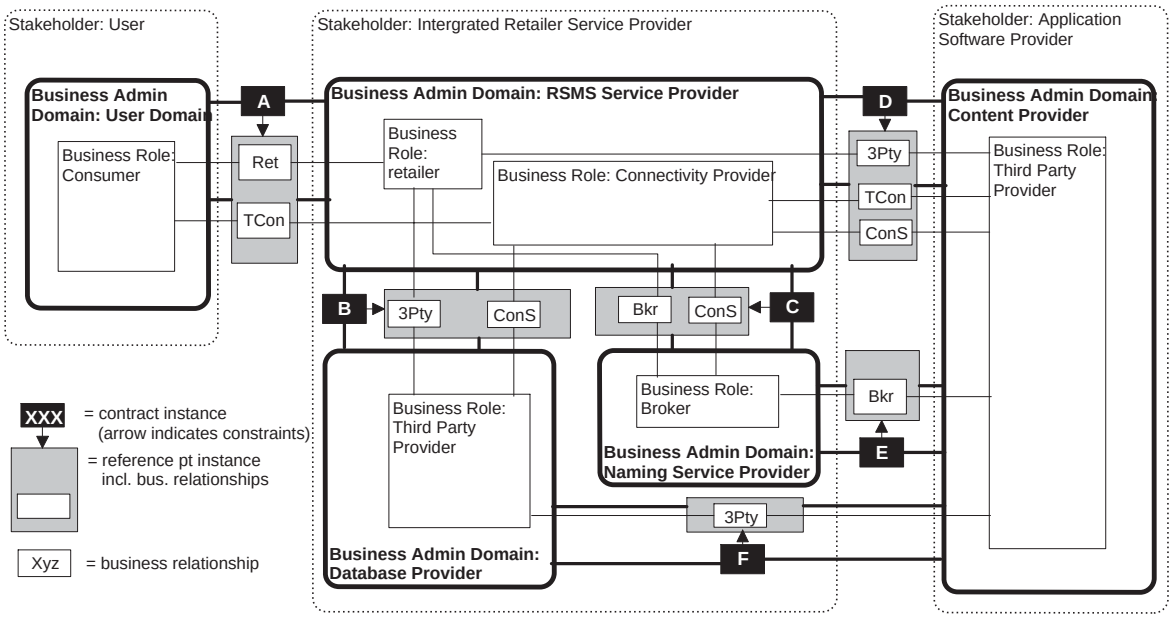


Figure 2.2: Business Relationship Instance Diagram

Figure 2.2 shows the business relationship instance of the SATINA RSMS service. The diagram shows three main stakeholders of the service, namely the User, the Integrated Retailer Service Provider and the Application Software Provider. On the top-left of the figure is the stakeholder-User, it has one business administration domain named the user domain. It can be thought of as the PC or the equipment that is under the user's control. The business role within this domain is the consumer.

The Integrated Retailer Service Provider has three different business administration domains namely the RSMS Service Provider, the Database Provider and the Naming Service Provider. The SATINA RSMS implementation considers the database and the naming service belonging to the same stakeholder. This provides a smooth transition path from non-TINA to TINA service, as a non-TINA service provider

would usually have its own database functions attached to the service. However, the database functions can be completely divided from the main service logic if necessary. A more generic database function, which acting as a third party service provider, can be provided to different service providers in this distributed process environment. Although the database is presented as a generic third party provider, currently no other service depends on it. Therefore, it is considered that the Integrated Retailer Service Provider has complete control of the database services.

The naming service is a simplified version of a broker function defined in the TINA business model. The purposes of the naming service are for the third party content providers to register themselves in the retailer domain and to enable the retailer to locate the content providers. Since this broker function is not a public service and is not supposed to be shared among different retailers, it is regarded that the Integrated Retailer Service Provider has complete control over those who can register in the naming service. However, this broker function can be shared among different retailers without modification of the implementation code.

Inside the RSMS Service Provider business administration domain, two roles are defined: one is the retailer and the other is the connectivity provider. Since the SATINA platform does not have an implemented instance of a separate, independent connectivity provider, it is considered that the network connection belongs to the retailer service provider who has control over the network connections.

The stakeholder Application Software Provider is on the right-hand side of figure 2.2. It has one business administration domain called Content Provider and it has a business role as a third party service provider.

Between different business administration domains, reference points and contracts are defined. The reference points provide a set of standard interfaces wherein objects can invoke operations on one another. Contracts provide constraints on which interfaces can be used and how it should be invoked.

The reference-points-instance bounded by contract A has two business relationship namely the Ret and TCon, which are defined in the TINA business model. It shows that the user has retailer relationship to the RSMS Service Provider and the user accesses the network though the TCon interfaces. The overall functionality and scope of the Ret reference are standard and can be found in [4]. The functionality of TCon is not defined in this project.

In contract-instance E and C, the Bkr reference point is applied. The overall functionality of the Bkr is to provide third party service providers interfaces to register and deregister their service, together with useful information about their services. It also provides interfaces for retailers or other brokers to retrieve the information about the registered third party service providers. As there is presently no document released by TINA consortium on the Bkr reference point, it is not TINA compliant.

The reference point instances constrained by contract instance B, D and F contain the 3Pty reference point. The SATINA implementation uses the Ret reference point as the basis for the 3Pty reference point, with added customised interfaces to support 3Pty operations. The design and implementation can be found in chapter 4 and in chapter 5. The main functions of the 3Pty reference point are authentication, starting the access and usage sessions. The reference point instances governed by contract instance B and F is related to the database, which provide interfaces of accessing and modifying the data stored in the database. The reference point between the RSMS Service Provider and the Content Provider offers interfaces to access the main ASP related functions, which is discussed in detail in later chapters.

2.6 Use Case Diagrams

Use case diagram is one of the many useful tools defined in UML to describe part of a complex system. The use case diagrams are employed to help illustrate the different roles and functions of the stakeholders defined in the TINA business model. Use case diagrams comprise of actors, use cases and their relationship within the system. The definitions [8] of an actor and a use case are the followings:

> **An actor** is a role of object or objects outside of a system that interacts directly with it as part of a coherent work unit (a use case). An Actor element characterises the role played by an outside object; one physical object may play several roles and therefore be modeled by several actors.

> **A user case** is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside interactors(called actors) together with actions performed by the system.

In order to explain different aspect of the system, three use case model packages are sketched, namely the *Service Usage*, *Service Maintenance* and *Third Party Sessions*.

2.6.1 Service Usage

The Service Usage use case model package presents the scenarios that are related to service usage issues. The model, which is shown in figure 2.3, contains three use cases: *Request Software List*, *Remote Execute* and *Log Usage Record*.

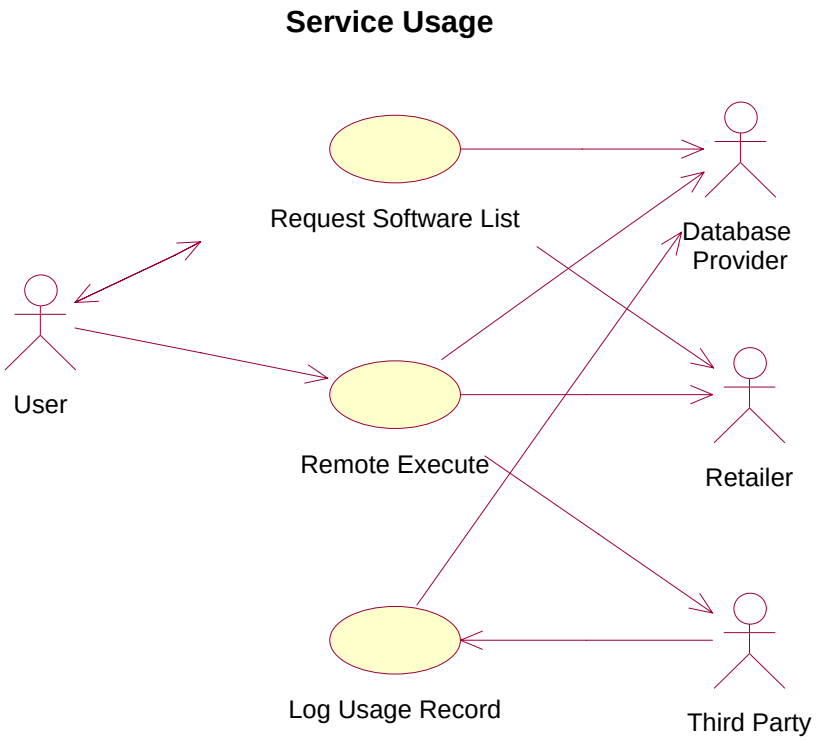


Figure 2.3: Service Usage Use Case Model Package

The use case Request Software List shows the situation that when the user queries the system for a list of available application software. It is not the interest of this plane to understand the detailed operations[1] in this use case, but the actors involved. As in figure 2.3, the actors involved in this use case are the user, the database provider and the retailer.

The use case Remote Execute reflects the situation that the user requests the desired application software to be executed in the remote server[2]. The actors involved in this use case are the user, the database provider, the retailer and the third party content provider.

The last use case that is defined in this package is Log Usage Record. This use case describes the procedure that the third party content provider logs a usage record

[1] See section 4.3.2 for details
[2] See section 4.3.2 for details

after the user has executed a particular application software. The third party content provider and the database provider are involved in this use case.

2.6.2 Service Maintenance

The Service Maintenance use case model package expresses the facts on the service maintenance issues. This model contains five use cases, namely *Check Service Mode*, *Modify Service Mode*, *Check Software List*, *Modify Software List* and *Check Usage Records*.

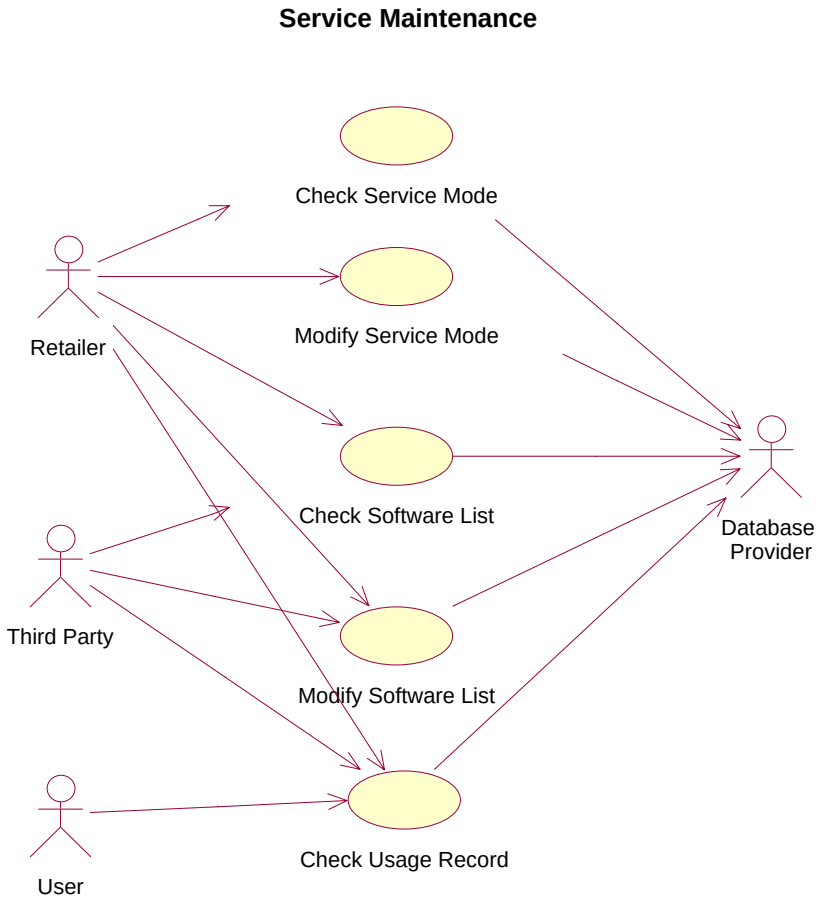


Figure 2.4: Service Maintenance Use Case Model Package

The use cases Check Service Mode and Modify Service Mode are only accessible by the retailer. They are used by the retailer to manipulate the available service modes that can be applied to all application software. Example service modes that can be applied to the software are "Remote Executable" and "local installation" mode. The retailer can add, remove or modify the available service modes.

The use cases Check Software List is accessible to the retailer and the third party. According to the policy defined in section 2.4, the retailer can read all available software in the central database, with detailed information about the individual application software in the list. The third party content provider can read all available software under its registration. Similar rules apply to the use case Modify Software List where the retailer or third party content provider can modify the list or modify the information associated to the individual application software.

The use case Check Usage Record is accessible to the retailer, the third party and the user. The only difference between the three parties is that the retailer can check all the records; the third party can only check the usage records that are related to it; the user can only check his own records.

2.6.3 Third Party Sessions

As the 3Pty reference point standard has not been released by TINA consortium, the 3Pty reference point and related functions are defined along side with the RSMS project. Figure 2.5 shows the use case model package that describes third party sessions.

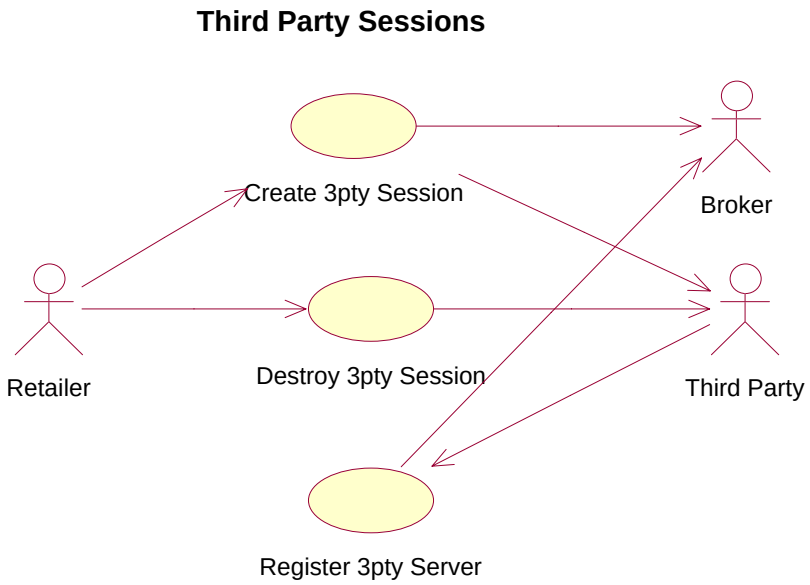


Figure 2.5: Third Party Sessions Use Case Model Package

This package includes three use cases namely *Create 3pty Sessions*[3], *Destroy 3pty*

[3]See section 4.3.2 for details

Sessions[4] and *Register 3pty Server*[5]. The first use case is related to the creation of a third party session, which involves the retailer, the broker and the third party. The second use case shows that when destroying the third party sessions, only retailer and third party service provider are involved. The third use case presents the situation when the third party content provider informs the naming service about its existence.

[4]See section 4.3.2 for details
[5]See section 4.3.1 for details

Chapter 3

Service Design: Definition Plane

3.1 Introduction

The second step of service development is the definition plane. In this plane, the business model is used as the input to construct an object-oriented description of the service and its environment, without concerns about the actual implementation of the objects. Therefore the objects are not computational objects, rather they are information objects. Consequently, the information model serves as the output of this plane. These information specifications are produced early in the analysis phase of system development since they define the information necessary for using the system and thereby define the context in which the system is going to be used. The information model generated corresponds to the information viewpoint of the RM-ODP standard.

3.2 The Information Model

The service information model describes the world of service in terms of all the pertinent actors and components. It contains descriptions of the service environment as well as a description of the components used to build the service and their relationship, using the modeling notions defined by object modeling technique (OMT).

Unlike computational specifications, the information model is aimed to collect the knowledge necessary to make appropriate use of a system or any part of the system. One does not required to understand how the system is implemented or what kind of technology is used in order to use the system appropriately. The abstraction

level that applies to an information specification may vary from the very abstract where distribution aspects are not considered, to very low levels where the DPE is involved. The information specification defined in this section only focus on the high level aspects of the system. To reduce the complexity of the information model, only objects that are directly relevant to the RSMS service are shown.

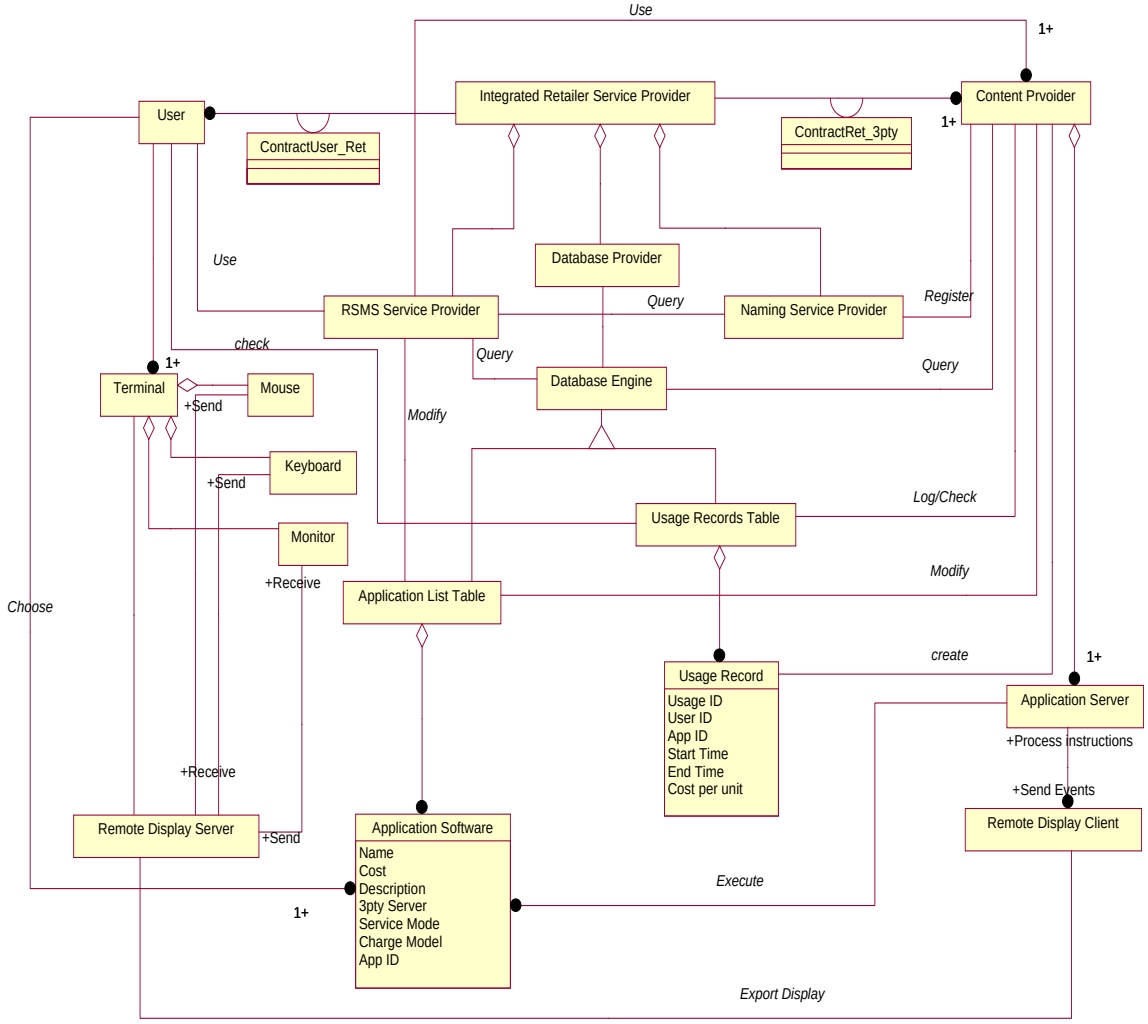


Figure 3.1: Information Model

The rectangular blocks shown in figure 3.1 are the information objects. Since this model is aimed at providing high level abstract information of the system, the information objects do not necessarily have one-to-one relationship to the computational objects that are described in chapter 4. The line that links two information objects represents the relationship between the two objects. Different relationships can be represented by using different shapes at the end of the lines. Brief explanations of the notation are provided later when necessary. However, detailed descriptions of the notation can be found in [8].

The information model shown in figure 3.1 is organised in such a way that the higher the object, the higher abstract level the object has. The topmost three objects are the stakeholders namely the user, the content provider and the Integrated Retailer Service Provider. To explain this information model in an organised fashion, the model is subdivided into three main partitions that are related to the three topmost objects.

3.2.1 The User Partition

The user partition concerns the information objects that reside in the user domain. These objects are highlighted in figure 3.2, which is the portion on the left-hand side of the main diagram shown in figure 3.1.

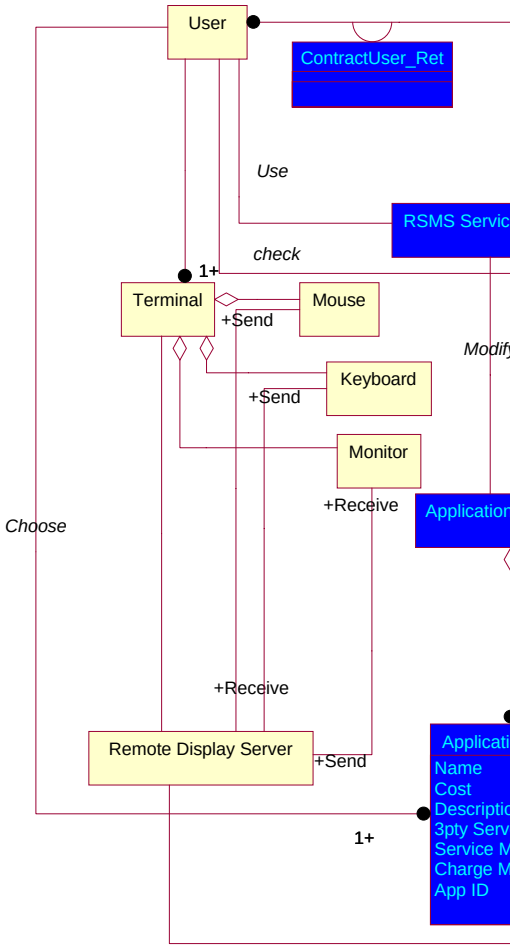


Figure 3.2: Information Model - User Partition Highlight

The six highlighted objects are the user, the terminal, mouse, keyboard, monitor

and the remote display server. The user object represents the role that the user takes part in the system. It has one line linking itself and the terminal object, which has one filled circle at the end of the line. This notation means the user can have one or more terminal.

The terminal has the mouse, keyboard and monitor objects attached. There are three diamonds at the end of the three lines. This diamond notation denotes that the terminal contains the mouse, keyboard and the monitor objects. These objects are the information models of the physical hardware attached to the system.

The terminal object also has a link connecting to the remote display server object. The remote display server object stands for the programming entity that runs in the user's terminal. It receives the inputs from the keyboard and mouse from the user's terminal and sends the signal, through a data stream connection, to the remote display client, which resides in the content provide domain, for processing. It also receives the screen output data from the remote display client. Subsequently, the remote display server draws the screen output on the monitor of the user's terminal.

3.2.2 The Content Provider Partition

The content provider partition, which is the portion that locates in the right-hand side of the main diagram, is shown in figure 3.3. The information objects belonging to this partition are highlighted in the figure.

The content provider partition consists of three information objects: the content provider, the application server and remote display client. The content provider object represents the business administration role that the application software provider participates in the service. Figure 3.3 shows that the content provider object can contain one or more application server objects.

The application server information object, similar to the terminal object in the user partition, represents the actual server that executes and processes the instructions given by the application software. It executes a particular application software when an execution request is issued from the content provider. The execution request is originated from the end user and forwarded to the content provider by the RSMS Service Provider.

The remote display client, similar to the remote display server mentioned in the previous section, represents a piece of software that is running under the application

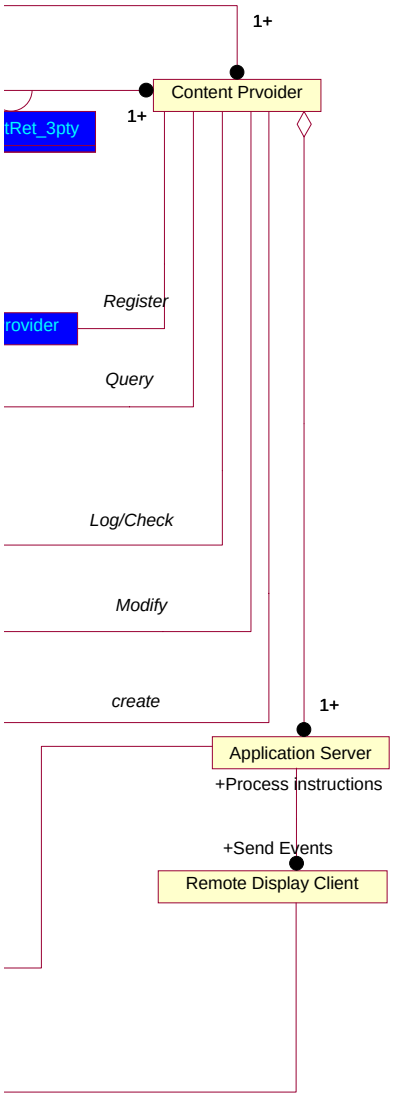


Figure 3.3: Information Model - Content Provider Partition Highlight

server object. There are two main functions of the remote display client. Firstly, it is used to receive all the user events, such as mouse click and key-down events, through a separated data stream between the remote display server and the remote display client. The received events are passed to the application server for processing. Secondly, the remote display client sends any updates data of the screen outputs, received from the application server, to the remote display server in the user partition. The update data is sent through the stream connection. This tight relationship between the application server and the remote display client is represented by placing a link between these two objects.

3.2.3 The Integrated Retailer Service Provider Partition

The Integrated Retailer Service Provider Partition is the middle part of the main information model diagram shown in figure 3.1. The objects that are assigned to this partition are highlighted in figure 3.4.

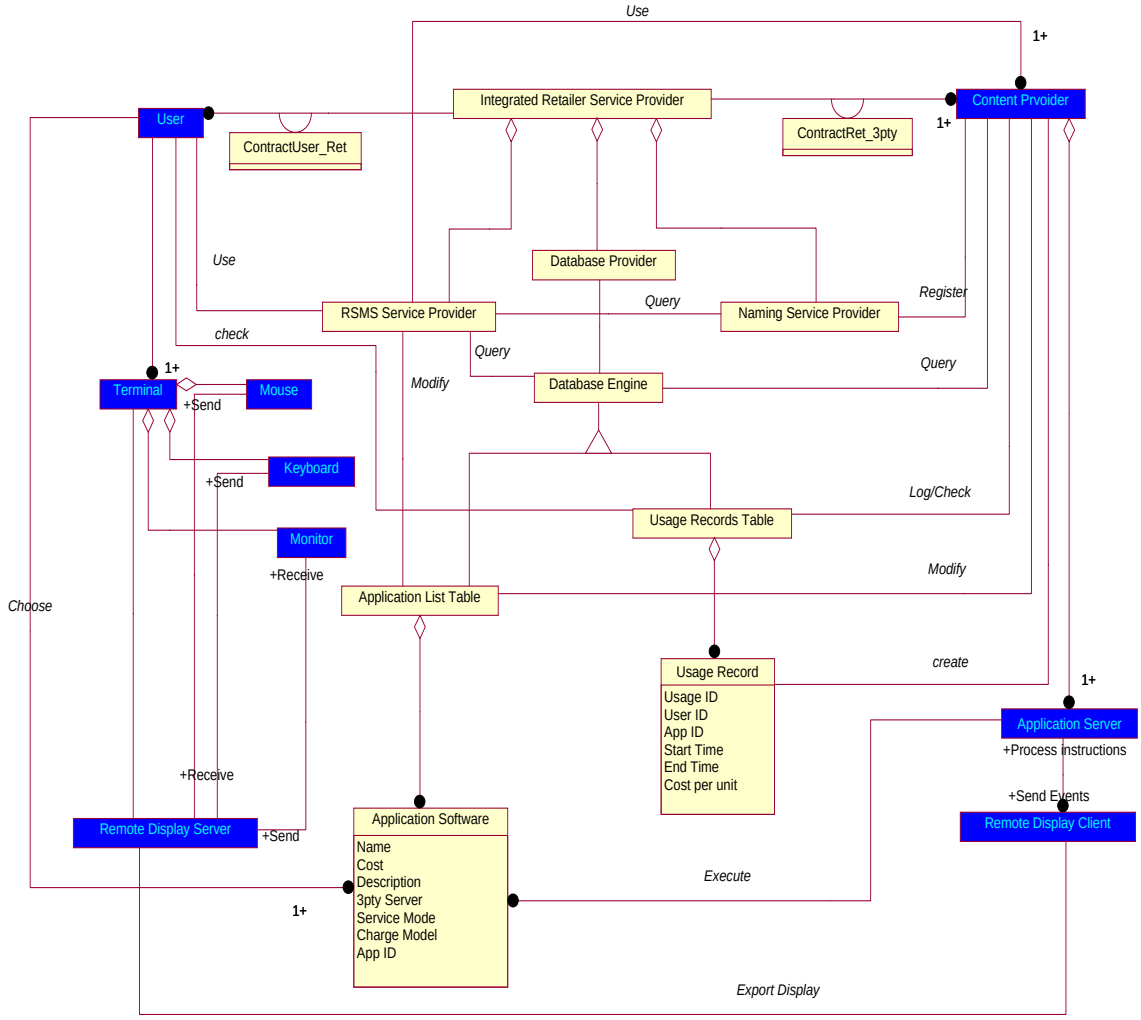


Figure 3.4: Information Model - Integrated Retailer Partition Highlight

The topmost object in this partition is the Integrated Retailer Service Provider. It represents the stakeholder in the retailer domain. The Integrated Retailer Service Provider object has links to three other information objects namely the RSMS Service Provider, the Database Provider and the Naming Service Provider. As described in the previous sections, the diamonds at the end of the links presents containment. Thus, the Integrated Retailer Service Provider object contains the three objects mentioned above. These three objects represent the roles that the Integrated Retailer Service Provider has to perform in the system.

The Database Provider Object

The Database Provider Object represents one of the few business administration domains in the stakeholder Integrated Retailer Service Provider. It has a link connecting to the database engine information object. The database engine object provides operation interfaces to other information object for accessing the database. The database engine contains two main tables, namely the application list table and usage record table. These two tables are represented in the model as two information objects.

The main function of the application list table is to store application software entries. An application software entry is presented as an application software object in the information model shown in figure 3.4. Each application software object provides descriptive information about the application software it represents. The application software object contains attributes such as the name of the application software, the cost, description of the application, the identity of the hosting third party provider, the service mode and the change model. The links between the user object, application software object and the application server objects show that the user can choose one or more application software to execute in the application server.

The purpose of the usage records table is to store usage record entries. The usage record entry is also presented in the model as usage record object. Every time a user has finished executing an application software, a new usage record is added into the usage records table. The usage record contains the information of who(the user ID), what(application software ID), when(start time and end time) and how much(cost) of a particular usage event. Through the database engine object, the user, the RSMS Service Provider and the Content Provider can check the usage records for billing or verification purposes. The relationships between the usage record table and the stakeholders can be seen from the links between these objects in the information model.

The RSMS Service Provider Object

The RSMS Service Provider object is another business administration domain that the Integrated Retailer Service Provider contains. It is the frontier object that serves the user. The users authenticate themselves by logging into the RSMS Service Provider domain and establishing access sessions and usage sessions[1]. The link

[1]See section 4.3.2 for details

between the RSMS Service Provider and the database engine comes from the fact that the RSMS Service Provider needs to query the database engine for a list of available application software. When a new service session starts or when the users indicate that they want to update the software list, the RSMS Service Provider contacts the database provider for a list of available software.

The relationship between the RSMS Service Provider object and the content provider object is very similar to the connection between the user and the RSMS Service Provider object. The RSMS Service Provider authenticates itself to the content provider and establishes both access sessions and service sessions just like the user. The user requests, such as request for software execution, are forwarded from the RSMS Service Provider to the content provider if the requests require the content provider to accomplish.

The Naming Service Provider Object

The last business administration domain that the Integrated Retailer Service Provider contains is the Naming Service Provider object. It represents the broker in the information whose main responsibility is to provide information about the third party providers to the retailer.

The Naming Service Provider object has two links, one to the RSMS Service Provider and the other to the content provider, representing the relationships with each other. For the content provider to announce its existence, the content provider registers itself with the Naming Service Provider by posting its contact reference to the Naming Service Provider. On the other hand, the Naming Service Provider answers the queries from the RSMS Service Provider about the existence of the content providers. If the queried content provider is found in the registry, its contact reference is returned back to the RSMS Service Provider.

The Contract Objects

The link between the user and the Integrated Retailer Service Provider, and the link between the Integrated Retailer Service Provider and the content provider have contract objects attached. The contract object points out that between the user and the Integrated Retailer Service Provider, there is a contract bounds to the relationship between the two objects. Similar situation applies to the Integrated Retailer Service Provider and the content provider. The contact objects show that

different users may apply different constrains to the relationship bound to the service provider and vice versa.

Chapter 4

Service Design: Design Plane

4.1 Introduction

In this chapter, the third plane of the system design process is presented. This plane is called the design plane which contains object-oriented description of the components of logic and data. These components are the inputs of the implementation (coding) of the software of the service. The output of this plane is the computational model of the service, which defines the units of programming called computational objects. This model maps to the computational viewpoint of the RM-ODP architecture.

In addition to the computational model, flowcharts and sequence diagrams of different events are used to illustrate the logic of interactions of the computational objects.

4.2 Computation Model

Whereas the information model provides the necessary knowledge of interacting appropriately with a system or subsystem, the computational model specifies the system itself. Objects identified in the information model may correspond to objects in the computational model, but this is not necessary to have one-to-one relationship between the information objects and the computational objects, which is the case of the RSMS.

The main components of the computational model are the computational objects

(CO). CO, which provides encapsulation of data and processing instructions, may contain several operational interfaces, which are the interaction points for other objects to access its data or processing instructions. It may also contain stream interfaces, which facilitate stream connections between itself and other COs. A stream can be regarded as a sequence of bits, which represents some information flow.

The TINA service architecture defines four generic sessions for any services, namely the access session, the service session, user session and the communication session. The effect of separating the sessions emerges on the computational model shown in figure 4.1. The computational objects above the horizontal-dotted line are access-related components where the objects under the horizontal-dotted line are usage-related components. The vertical-dotted lines separate the stakeholder domains, which are related to the information objects defined in the chapter 3. Since the vertical-dotted lines separate different domains, they also represent the reference points defined in the business model.

The access-related components provide a framework for secure and personalized access to services and a framework for mobility support. The provider agent (PA) (2 in figure 4.1) and the user agent (UA)(4) interact within a secure, trusted relationship between the user and the integrated retailer service provider. They support authorization, authentication and customisation of the service access, and provide a secure mechanism for starting and joining service sessions.

Usage-related components provide a framework for defining services, which can be accessed and managed across multiple domains. The service session manager (SSM) (6 in figure 4.1), user service session manager (USM)(7) and the composer usage session manager (compUSM)(8) are instantiated by the service factory (SF)(5). The compUSM is a generic component that is served as an interface to the third-party service provider. Since the USM, the compUSM and the SSM are created by the SF each time a new service session starts, they are dynamic objects with life span the same as the service session. If another ASP retailer were involved, the PeerUSM[1] would be used to interface between different retailers. The USM, compUSM and PeerUSM are inherited from the member usage session manager (MUSM) object defined by [2].

Service specific components are a subset of the usage-related components, which provide the functionalities that are specific to a particular service. For example, the RSMS Core object (15), Retailer Admin GUI object (22) and 3pty Admin GUI

[1]PeerUSM is currently not defined in this project

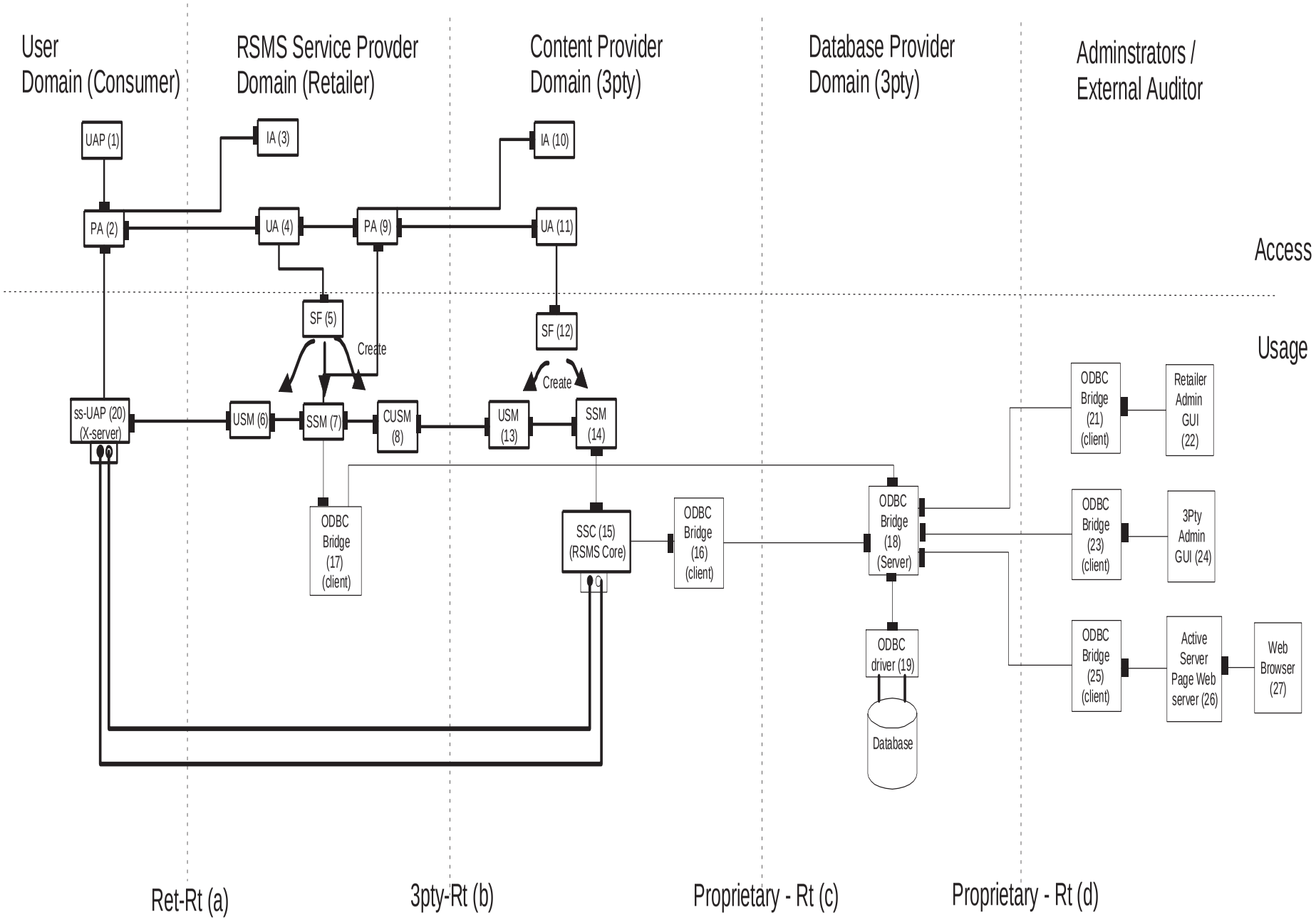


Figure 4.1: Computational Model

object (24) are specifically designed for RSMS use. The ODBC Bridge Client and Server objects (16,17,18,21,23,25) are for the exclusive use of data depository. All these components together constitute the service logic. The service support component (SSC) represents special resources controlled by the service session. In this case, the RSMS Core object (15) is regarded as an inherited instance of the SSC object. The RSMS Core object contains a streaming interface to communicate with the client's service specific User Application (ssUAP)(20). The RSMS Core object is a dynamic object as it is only created when the third party service session is established. The database service-related objects, such as the ODBC Bridge client and ODBC Bridge server, are static objects. They only have a single instance, which services requests from different third party service sessions.

The interfaces of the retailer reference point (a in figure 4.1) between the consumer and the RSMS retailer are implemented using the standard retailer reference points defined by [4]. However, the reference points between RSMS third party and the database depository (c) are currently implemented using interfaces not specified by TINA.

Since the TINA third party reference point have not been released yet, the third party reference point (b) implemented in SATINA is basically a cascaded retailer reference point with SATINA specific extensions to support third-party-related functions. The standard retailer reference point components are only discussed briefly in this document, as detailed reference can be found in [4]. On the other hand, the objects that support the SATINA specific extension are discussed in detail later in this chapter and the following chapters.

4.3 Service Logic

This section discusses both the service logic of the RSMS service and various session models that are involved in the service logic. To aid explaining the service logic clearly, this session is subdivided into three main subsections namely the Pre-Service Events, the Main Service Events and the Auxiliary Service Events. These subsections map the use cases, which are defined in chapter 2, into different events or activities. These events are described in terms of the interactions between the computational objects. All of these events comprise the service logic of RSMS.

4.3.1 Pre-Service Events

The pre-service events are the activities that are the prerequisites before the service could start. Although they are important, they are not directly related to the service logic.

Content Provider Registration

The naming service server commences its service before the RSMS service provider (retailer). This enables the third party PA (9 in figure 4.1), which represents the content provider (third party), to register itself with the naming service before the retailer starts the service. Each of the content providers must have a different third party PA representing themselves in the retailer domain. The third party PA contains the object reference of the IA(10) of the content provider it belongs. The reason for this registration is that the retailer needs to know which third party content providers are available when the service starts. The retailer accomplished this by querying the naming service through the specific interfaces, which can be considered as the simplified, proprietary broker reference points. Since the third party PA is designed as a completely service generic component, it is located above the horizontal-dotted line in the "Access" region in figure 4.1.

In addition to the registration of the third party PA, the initial agent (IA) (3, 10), user agent (UA) (4, 11) and service factory (SF) (5, 12) in the retailer domain and third party domain must be initialised before an access session starts.

Exporting the Available Software List

Initially the database depository does not have any application software entries in the database. The third party content providers therefore have to export or register the application software that will be hosted for the consumer to use. This is done through the specific designed third party administration GUI (24 in figure 4.1), the ODBC Bridges (23, 18) and the proprietary reference point (d). The retailer has the right[2] to modify the available software entries at later stage using the retailer administration GUI (22). The exporting of the available software list has to be done before the retailer service starts because the retailer needs the list to present to the consumer once the service has started.

[2]Refer to the policies defined in chapter 2

4.3.2 Main Service Events

This section contains the events directly involved in the service delivery. The use cases defined in chapter 3 are served as the inputs of the event design. Contrary to the use case diagrams, which shows the service events from the stakeholder's point of view, figure 4.2 shows the logic flow sequence of the events.

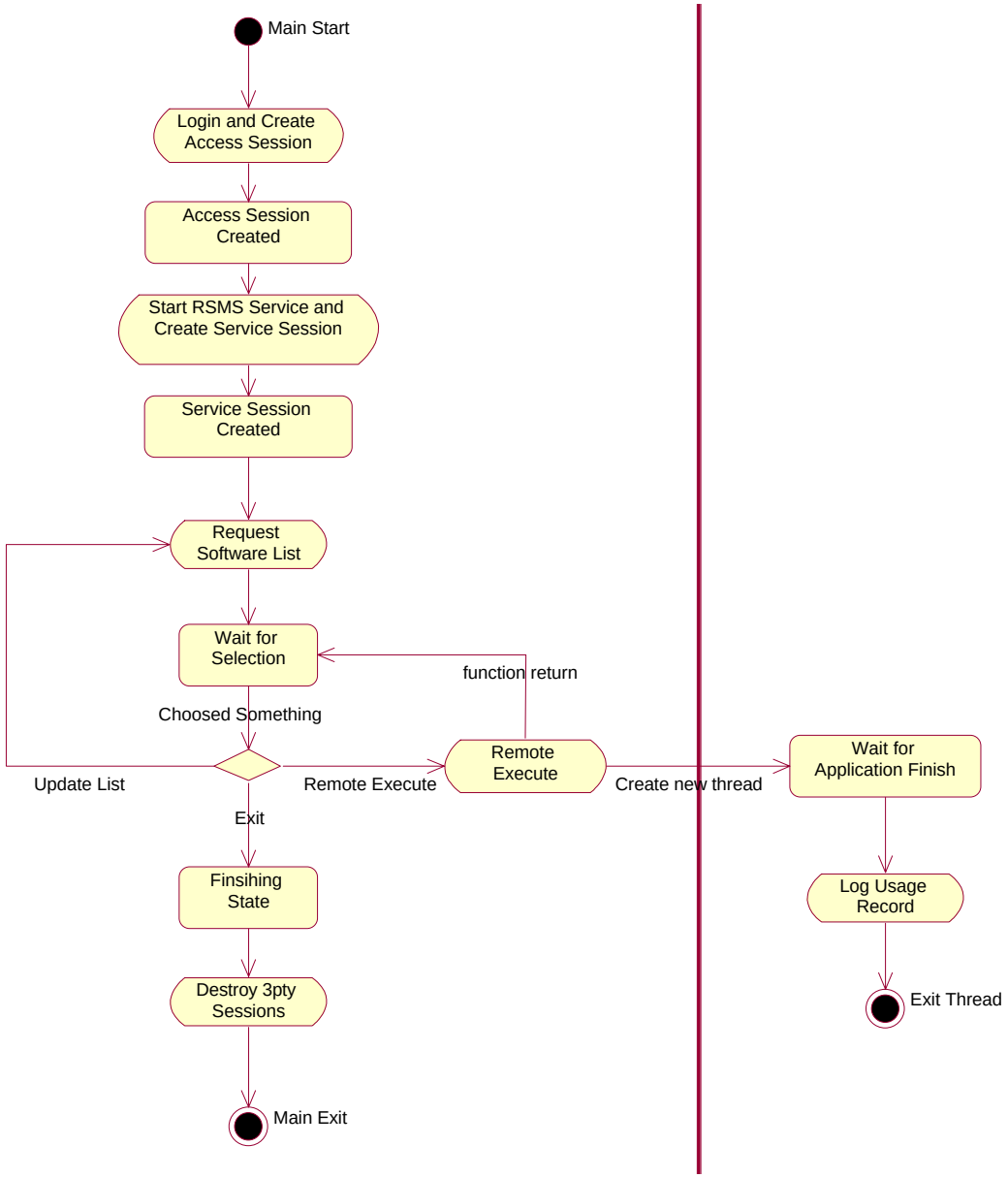


Figure 4.2: Main Service Flow

The rounded blocks of the flowchart are the events (activities) and the rectangular blocks are the states. An overall description of the flowchart in figure 4.2 is given below and the detailed explanation of the events follows.

The first event is the login event, which enables the users to authenticate themselves.

After authentication, the access session between the consumer and the retailer is created. Once the access session has been created, the user is presented with a list of subscribed services for him to choose. The retailer service session is then created once the user has chosen to start the RSMS [3]. Thereafter the retailer requests a list of available application from the database provider and forwards it to the user. The user can then select to exit the service, update the received list or execute one of the application software entries in the list. If the user chooses to execute an application software, the retailer starts the *Remote Execute* event, which creates a new thread that issues an execution order to the third party content provider. The new thread continues governing the execution of the application software, even if the user has exited the service and the main thread no longer exists. After the user has finished using the application software, the new thread will create a new usage record entry to the database provider.

While the new thread is executing the application software, the main thread returns to the *Wait for Selection* state loop. The user can therefore choose to start another application simultaneously with the running application. If the user chooses to exit the service, the retailer enters the *Finishing* state and destroys all third party service sessions and retailer service sessions.

Create Access Session Event

When the users want to start a service, they start the client package which will request the users to enter their login and their password. The client package contains both the user application (UAP)(1 in figure 4.1) and PA (2). The authentication information is then sent via the PA to the IA (3) and UA (4) in the retailer domain to create a retailer access session[4].

Similar to the consumer, the retailer has to authenticate itself to the third party content provider with the procedure described above to establish a third-party access session. While the database depository components (16,17,18,19,21,23,25) are not fully implemented as TINA objects, the RSMS Core object (15) is pre-programmed to hold the reference of the database depository. Thus, the RSMS Core object is able to access the database directly without authentication and without creating an access session.

[3] To reduce irrelevant branches of the logic flow, it is assumed that the user chooses to start the RSMS only.

[4] Detailed event traces, object interactions, and interfaces involved have been defined by TINA consortium which can be found in [2].

To provide the service, at least two access sessions are needed. One access session connects between the consumer and the retailer; another access session connects between the retailer and the third party. If more then one content provider is involved, the retailer might create multiple access sessions to logon to different third party domains. These access sessions are generic to different kind of services. E.g. Video on Demand service, Voice over IP service and ASP can share the same access sessions. A single third party service provider can offer multiple services shared by a single access session. However, the retailer has to create separate access sessions to connect to different third parties since the retailer might use different identity to logon to different third parties. Figure 4.3 shows the relationships between the sessions and domains.

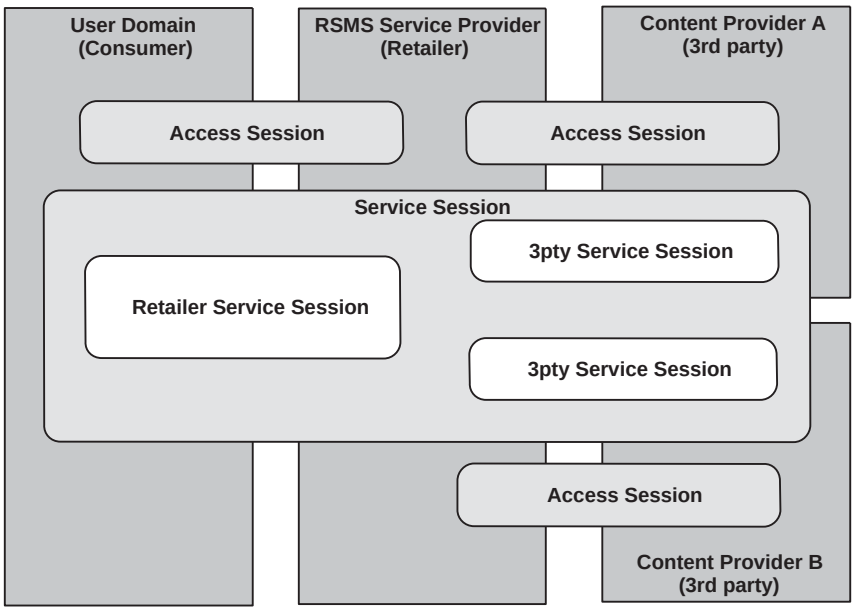


Figure 4.3: Sessions and Domains

Create Service Session Event

After the access session has been established between the client and the retailer, the client is then prompted to choose a service. As soon as the RSMS service has been chosen, the service factory (SF) (5 in figure 4.1) dynamically creates two service specific components, namely the user session manager (USM) and the service session manager (SSM). Since all services can share the same access session, the SF will have to instantiate the correct service component according to the service. This is done by assigning a different service ID to different services. A new USM (6) and SSM (7) will be created with each new service session using the corresponding service ID. Detailed event traces of creating the service session are defined in [2].

Each time the retailer starts a third party service session, a new composer user session manager (compUSM)(8) will be created by the SF. Therefore, there will be a unique USM, a unique SSM and more then one compUSMs within the same retailer service session. The compUSM is unique within a single third party service session, as one retailer service session can have multiple third party service sessions. The retailer service session starts as soon as the USM (6) and the SSM (7) have been created. Thereafter the ssUAP (20) can communicates with the USM (6).

To delivery the service to the consumer, one retailer service session and at least one third-party service sessions are required. If more then one third party content providers are involved, multiple third party service sessions can be created, each corresponds to a particular content provider. The retailer service sessions and the third party service session(s) can be considered as the sub-sessions which comprise of the main service session. (Refer to Figure 4.3)

Request Software List Event

This event can be mapped to the use case *Request Software List* shown in the figure 2.3 in Chapter 2. Whenever the user requests a list of available software, the ssUAP (20 in figure 4.1) sends the requests to the interface `i_PartyAppSoftManager`[5] on the retailer USM (6) by calling the function `Request_Software_List()`. Next, the retailer USM (6) forwards the request to interface `i_ProviderAppSoftwareManager` on the retailer SSM (7), which holds the reference of the database provider. The retailer SSM then queries the central database in the database provider for a list of currently available application software on the third party content providers. The return parameters of the function `Request_Software_List()` include a programming object type called `AppSoftList`, which is a list that contains an sequence or an array of `AppSoft` programming object. An `AppSoft` object contains the information of an application software package, such as the name, cost and location of the third party service provider etc. Once the list has returned to the retailer SSM (7), it may apply the necessary filtering operations on the result and may send a modified list of available applications to ssUAP (20) for the consumer to choose. Figure 4.4 shows the event traces of the Request Software List Event.

[5]See 5.3.3 for details

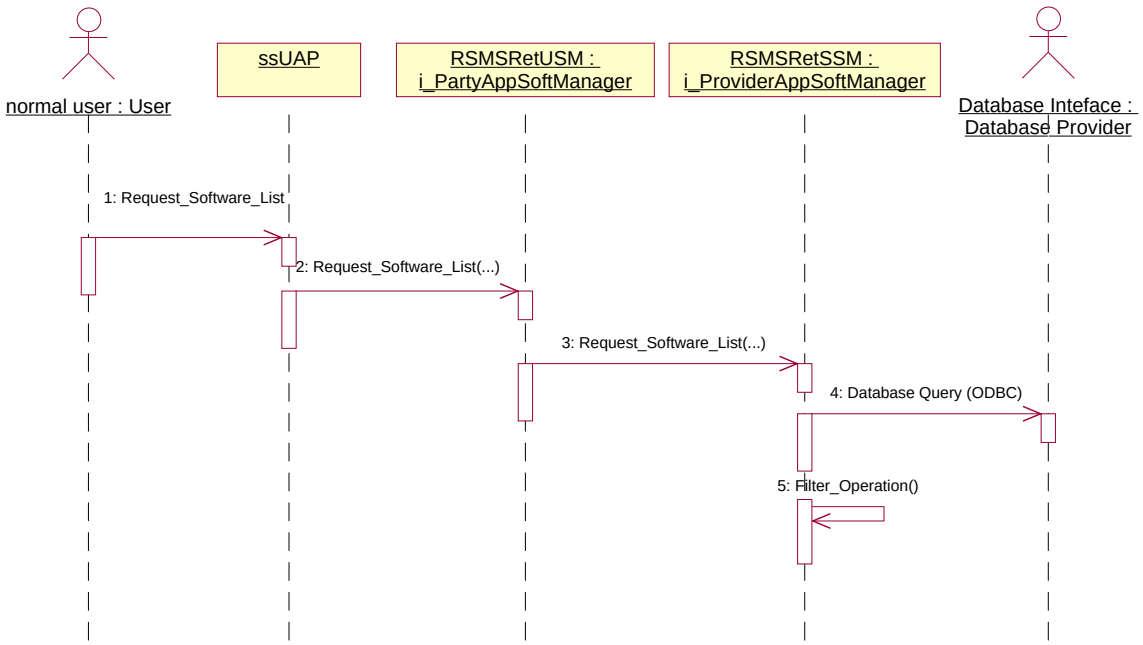


Figure 4.4: Request Software List

Remote Execute Event

Due to the complexity of the Remote Execute Event, it is subdivided into smaller events, namely the *Retrieve Host Server Information Event*, *Search for Established Third Party Service Session Event*, *Create Third Party Sessions Event* and *Execute Software in New Thread Event*. A flowchart (shown in figure 4.5) and an event trace (shown in figure A.1) are used to describe the logical sequences of these events and the object interfaces involved in these events.

Retrieve Host Server Information Event

When the user issues an order to execute a particular hosted application, the ssUAP immediately invokes the function `Remote_Execute_AppSoft()` on the retailer USM (6 in figure 4.1). After that, the retailer USM forwards the request to the retailer SSM (7). Once the retailer SSM has received the request, it will check which third party content provider should be contacted. This is done by examining the property *Server ID* contained in the `AppSoft` object, which is one of the input parameters of the function `Remote_Execute_AppSoft()`. The retailer SSM can then use this information to search and contact the correct third party content provider.

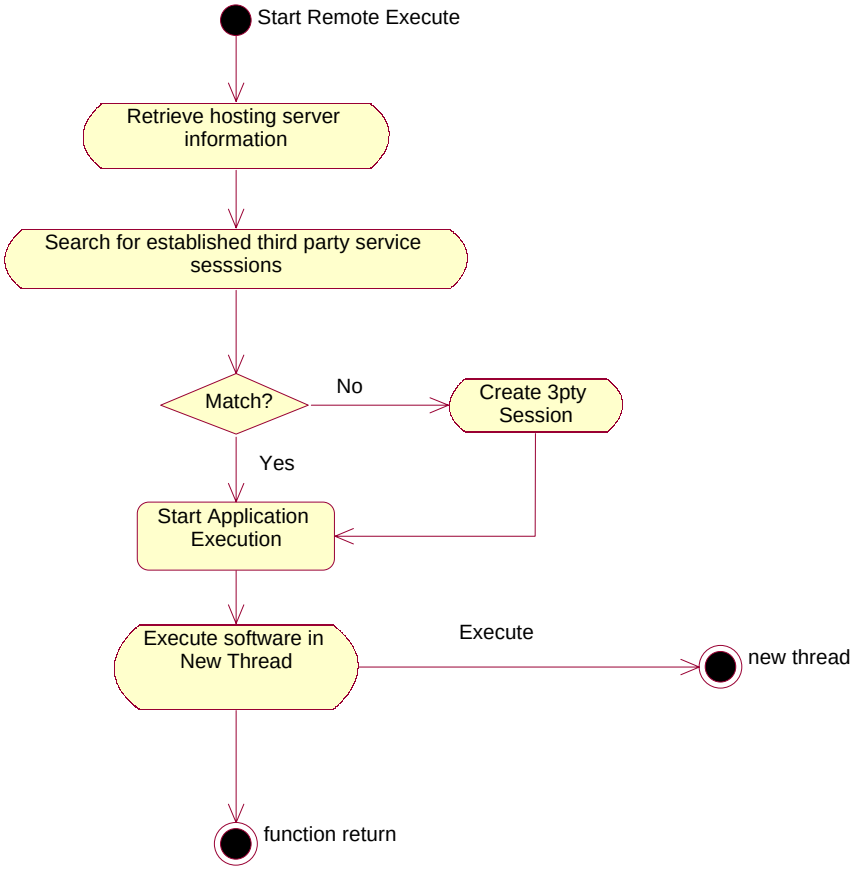


Figure 4.5: Remote Execution Flowchart

Search for Established Third Party Service Sessions Event

To process the software execution requests from the consumer, the retailer SSM (7) has to use the service rendered by the third party content provider. Therefore, a third party service session between the retailer and third party is required. The retailer SSM (7) internally holds a table of the currently established third party service sessions, together with detailed information about the sessions, such as the session ID, the third party Server ID and the object reference of the third party PA. As soon as the retailer SSM has interpreted the *Server ID* value, it checks all the entries in that table to see if the service session with the particular third party server, which hosts the requested application software, has been created or not. If the session is not present in the table, the retailer SSM will proceed to call the function `start_3pty_session` on the interface `i_Provider3rdpartyReq` to establish a service session between the retailer and the requested third party content provider.

Create Third Party Sessions Event

If the required third party service session has not been created, the retailer SSM (7) will instruct the naming service to look for the agent of the particular third party content provider. The result from the naming service is an object reference pointing to the corresponding third party PA (9). By the time the retailer SSM queries the naming service, the third party service provider should have already registered with the naming server[6]. If it is not case, the naming service will not be able to find the corresponding third party PA (9) and a *Service Not Available* exception will be thrown. As soon as the retailer SSM has received the third party PA reference, it requests the third party PA to start the *access session* with the content provider which the third party PA represents[7]. After the access session between the retailer and third party content provider has been created, the third party PA will obtain the content provider's service by starting a *third party service session* with that content provider. These two steps (creating the access session and third party service sessions) are done by invoking the `Start_3pty_Service` function on the interface `i_thirdPartySupport`, which is one of the interfaces on the third party PA, using the necessary information, such as the third party login, password and service ID, as the function parameters.

The third party PA summits the authentication information to the content provider and it will try to start an access session between the retailer and the content provider[8]. If the access session has been created successfully, the third party PA will start the third party service session. To setup the third party service sessions, the third party SF (12) needs to instantiate the third party USM (13), third party SSM (14), and the RSMS core object (15). The RSMS core object is a service specific object which contains all the intelligence needed to execute the service instructions. The object references of the interfaces of the newly instantiated third party USM (13) are then passed all the way back to the retailer SSM (7) as the return parameter `sessionInfo` of the function `Start_3pty_Service`[9]. Thereafter the retailer SSM instructs the retailer SF (5) to instantiate a new compUSM (8) for the new third party service session. The returned object references from the third party USM (13) are also passed when instantiating the new compUSM. After the instantiation of the compUSM, the retailer SF returns the object reference of the newly instantiated compUSM to the retailer SSM. The compUSM will then acting

[6] Discussed in section 4.3.1 in this chapter

[7] Each content provider has its own representative third party PA in the retailer domain, its corresponding PA must contain the reference to contact the content provider's IA.

[8] Detailed operations similar to the one defined in [2].

[9] See section 5.3.6

as a proxy for the retailer to communicate with the third party. After the new compUSM is successfully instantiated, the setup processes of a new third party service session are completed.

Next, the retailer SSM (7) appends the newly created third party service session into its internal *third party sessions table*. Information regarding the new session, such as the object reference of the instantiated compUSM, is stored in the table. If the client later requests to execute an application software, which its hosting third party service provider is already in the third party sessions table, the retailer SSM will contact the corresponding compUSM directly, instead of creating a new access session and a new third party service session again. Figure A.2 shows the sequence diagram of this event.

Execute Software in New Thread Event

After the retailer SSM (7 in figure 4.1) has received the reference of the correct compUSM (8), the retailer will issue a request to execute the software to the compUSM, using the function `Remote_Execute_AppSoft()` on the interface `i_CUSMAppSoftManager`. Once compUSM has received the function call, it will forward the request to the corresponding third party SSM (14) through the third party USM (13). If the instruction is approved by the third party SSM (14), the third party SSM will instruct the RSMS core object (15) to execute the requested application by invoking the function `Remote_Execute_AppSoft()` on the interface `i_RSMSCoreExecManager`. The RSMS core object carries out the order by forking itself into the main thread and a new thread, which is a new operating system kernel process. The new kernel process actually handles the execution of the requested application. The main thread then returns to the retailer SSM (7) waiting to receive new service instructions.

A stream is established between the requested application, which is running by the new thread of RSMS core object (5), acting as the X-client, and the client ssUAP (20) acting as the X-server. This bi-directional stream carries the user inputs data from the ssUAP to the RSMS core object in one way. In the other way it transfers the screen output data from the RSMS core object to the ssUAP[10].

[10] X-protocol is used in the TCP stream for communication between the client and the hosted application, see the implementation plane

Log Usage Record Event

The RSMS core object (15) logs all user activities, e.g. information like whom, which application, when and duration, into the database depository (19). The new thread, which is described in the previous section, is also responsible for logging the usage record, in addition to the execution of the application software. The logging of the usage events happens whenever the user has finished and exited the application software.

The new thread is running in parallel to the main service thread. As a result, the service will certainly be recorded as soon as the user stops running the remote application, even if the user has exited the TINA RSMS service (main thread) while the user is still using the remote application.

Destroy Third Party Sessions Event

The Destroy Third Party Sessions event describes the processes involved when the user exits the RSMS service. The purposes of these processes for are the followings:

1. To clean up the created objects to free system resources
2. To shut down the third party access sessions and third party service sessions, for security reasons, after the service is finished.

Figure A.3 in the appendix shows the sequence diagram of destroying the third party sessions. When the user gives an indication to exit the system (e.g. clicking the EXIT button on the GUI), the ssUAP (20 in figure 4.1) asks the retailer USM (6) to end the sessions by invoking the function `endSessionReq()` on the interface `i_ProviderBasicReq`. The signal is then forwarded to the retailer SSM (7) where it calls the function `End_3pty_Service()` to terminate all existing third party sessions. The `End_3pty_Service()` function checks the internal third party sessions table and sends the function `endSessionReq()` to all the compUSM (8) objects that are listed on the table. Each compUSM then forwards the `endSessionReq()` to its corresponding third party USM (13), which in turn forwards the message to the third party SSM (14) and then to the RSMS core object (15). As soon as the RSMS Core object received the function call, it destroys itself and frees the resources it is holding on the system. The function then returns to the third party SSM and it starts freeing the system resource it holds. After that the function returns to the

third party USM and then the compUSM, they in turn destroy themselves in the similar fashion. When all these objects are destroyed, one third party session ends. Next, the retailer SSM begins to end the next third party session, if it exists in the third party sessions table. This continues until all the third party sessions have been destroyed.

After all the third party sessions listed in the third party sessions table hadended, the retailer SSM (7) destroys itself and returns the `endSessionReq()` function to the retailer USM (6). The retailer USM in turn destroys itself and the retailer service session is considered closed. Once the objects have been destroyed, their object references become invalid. Thus, unauthorised use of the references by other objects will not be possible.

4.3.3 Auxiliary Service Events

The auxiliary service events include all the activities that are not directly related to the service usage. It is highly related to the service maintenance use case model described in figure 2.4 in chapter 2. There are two main categories of events discussed in this section, namely *the Application Software Database Management Category* and *the Usage Record Query Category*. The Application Software Database Management Category describe the activities involved in modifying the application software database, which include the use cases *Check Service Mode*, *Modify Service Mode*, *Check Software List* and *Modify Software List* defined in figure 2.4. The Usage Record Query Category maps to the use case *Check Usage Records*.

Application Software Database Management

Alongside with all the TINA objects and components, a computational object called the Retailer Admin GUI (22 in figure 4.1) is included in the system to perform administration tasks of the application software database for the retailer. This object provides a graphical user interface that is capable of adding, removing and querying any hosted applications in the database (19). The service modes[11] and charge models can also be modified and reviewed using the same GUI. Figure 4.6 shows the event traces of modifying a software entry in the database.

[11] Service modes are the forms of service that are delivered to the client side. Currently there is only one service mode *Remote Execution*, but it can be expanded to include services such as *Local Installation and Management* of a particular software.

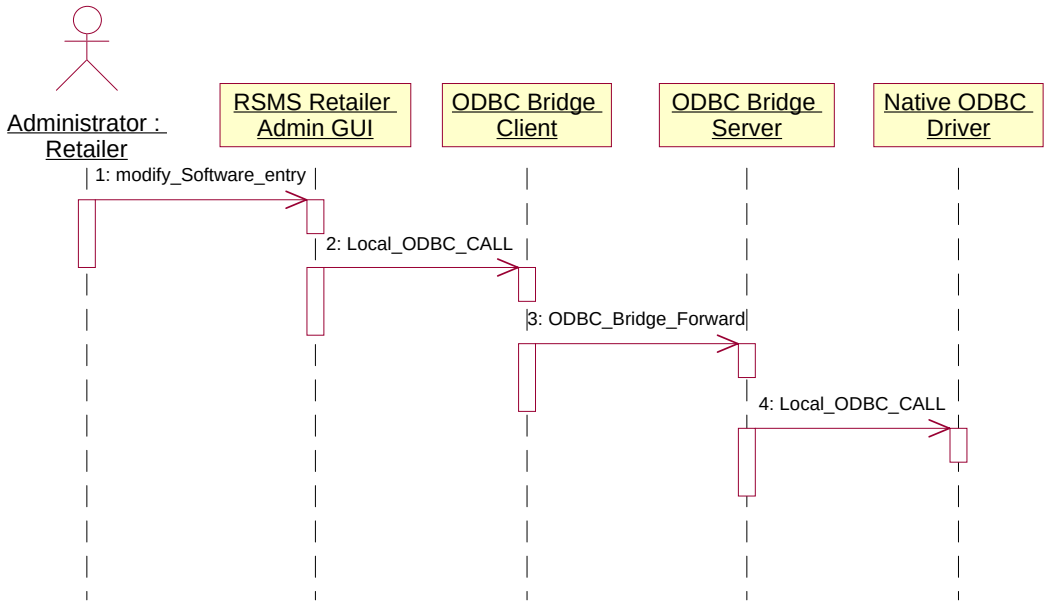


Figure 4.6: Modify Software Entry

When the administrator adds, removes, or modifies a software entry in the GUI, the GUI generates a SQL statement, which describes the modification, is sent to the local ODBC Bridge client through the local OS system call. The ODBC Bridge client contacts the remote ODBC Bridge Server using the proprietary reference point (d in figure 4.1). The ODBC Bridge Server then forwards the SQL statement to the local ODBC driver, which has the ability to access and modify the database according to the SQL statement.

Each of the third party content providers is provided with a component called the 3pty Admin GUI (24), which is the same as the retailer's administrative interface except the restriction of the access to database entries of other third party service providers. This third party administrative interface is designed for the third party service providers to add or update their own available application software on the central database depository, without interfering other third party service providers' data.

Usage Record Query

The retailer and the third party can check the usage records using the GUI components discussed in the previous section. The main different between these two cases is that the retailer can query all the usage records while the third party can only see the usage records related to their services. The usage record query goes through

similar process which has been explained in figure 4.6.

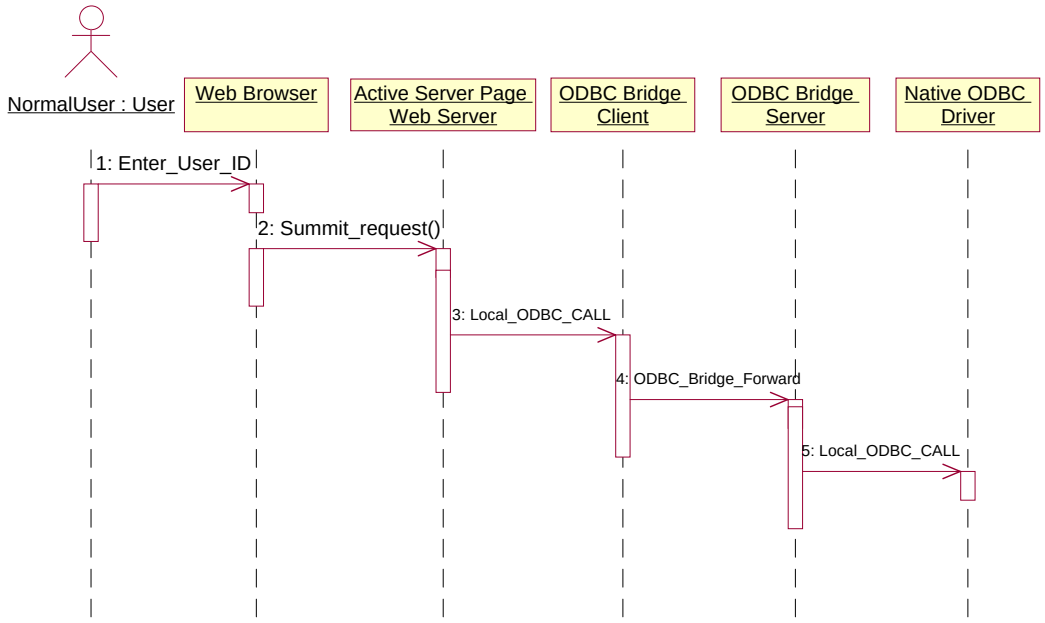


Figure 4.7: Usage Record Query by User

Besides the administration GUIs, the system includes a web interface (26 in figure 4.1) for the users to query their usage on the RSMS service. The users have to use a normal WWW browser (27) to visit the specified web page. The web page then prompts the users to enter their user ID. Once the users have entered the correct ID, the web server (26) executes a script that issues an SQL statement to the ODBC Bridge client (23). The request is then forwarded by the ODBC Bridge server (18) to the database provider (19). The matched records are retrieved and the usage information is passed back to the web server (26), again through the ODBC Bridges. After that the web server converts all the usage information received, such as the starting time, ending time, application name, cost, etc, into a normal web page and returns the page back to the client web browser. Figure 4.7 shows the sequence diagram of the user querying the usage records through the WWW interface.

Chapter 5

Service Implementation: Implementation Plane

5.1 Introduction

So far all the design issues discussed only involve in high level design matters, such as the roles of stakeholders, relationships between objects, functions of different object etc. In this chapter, focus is shifted to the real implementation concerns of the system, such as how objects communicate with each other and how to deploy objects into the system. The development phase described in this chapter is the implementation plane, which corresponds to the engineering viewpoint of the RM-ODP standard. This plane contains descriptions of the deployable software modules that comprise the service. The outputs of the implementation plane are the *engineering model* and the *implementation code*.

5.2 Engineering model

The TINA engineering modelling concepts[6] describe how to structure and deploy the application components in order to execute them on the infrastructure. The infrastructure defined in TINA is the Distributed Processing Environment (DPE), which hides the complex details of the mechanisms required for interaction between service components that are spread over a distributed heterogeneous system. The model for the distributed environment separates three main entities: TINA applications, DPE nodes, and the kernel Transport Network.

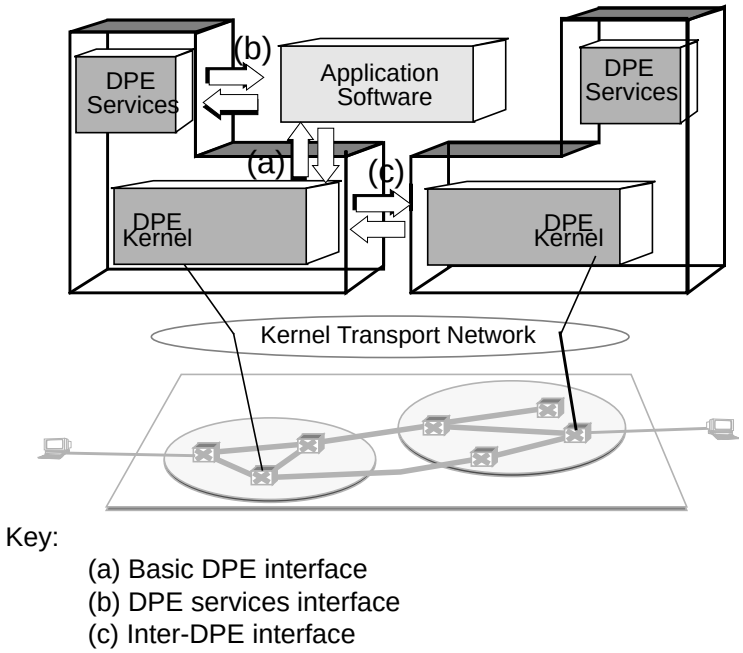


Figure 5.1: DPE Architecture

The DPE provides location transparency and basic services, such as object lifecycle management service, to engineering objects (eCO), which are the basic unit of the service components in the engineering model. The DPE makes use of the Kernel Transport Network to exchange discrete messages between the engineering objects. On the other hand, the Transport Network is used to establish stream connections between objects, for streaming of continuous data such as voice and video.

The location transparency characteristic of the TINA DPE enables object interactions independently of their location. It hides from the application components that they are executing in different computing environments in different DPE Node. Thus, the service designer does not need to consider the locations of the service components.

The TINA DPE is formed by a number of DPE nodes, which are the devices running the TINA DPE platform software on top of their native computing environment. A DPE node may contain one or more capsules. Within the capsules, there are clusters, which define groups of co-located engineering objects. In the current implementation of the RSMS, each eCO simply belongs to a single cluster, and each cluster belongs to a capsule. However, many capsules can co-exist in the same DPE node. Figure 5.2 shows the structure of a DPE node.

The computational objects (CO), which are defined in the previous chapter, are

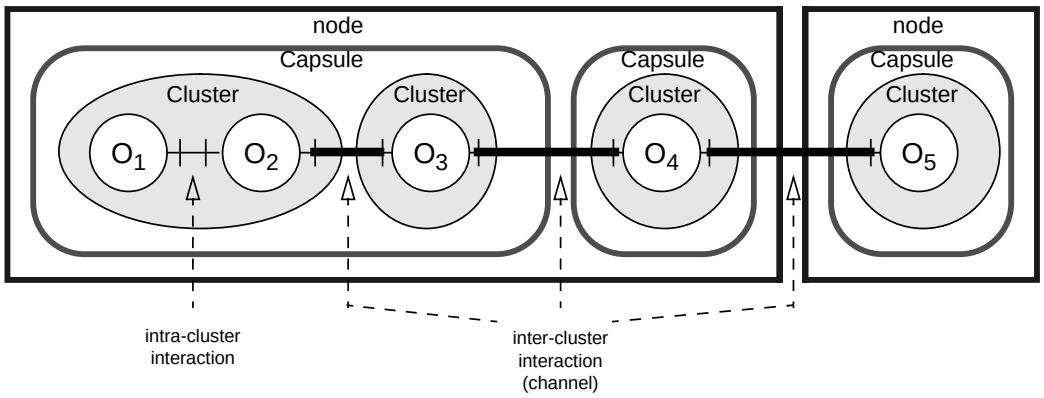


Figure 5.2: Node, Capsules and Clusters

mapped into the engineering objects in this implementation plane. The eCOs are then programmed or coded according to the defined parameters and behaviours. The CO has one-to-one relationships to the eCO template where the interface parameter types may be changed from the computational objects to the engineering objects, but the behaviours are identical. The eCO template can be instantiated into a particular instance of an eCO. For example, a new SSM eCO is instantiated every time a new service session is created. Figure 5.3 shows the relationship between the CO and the eCO.

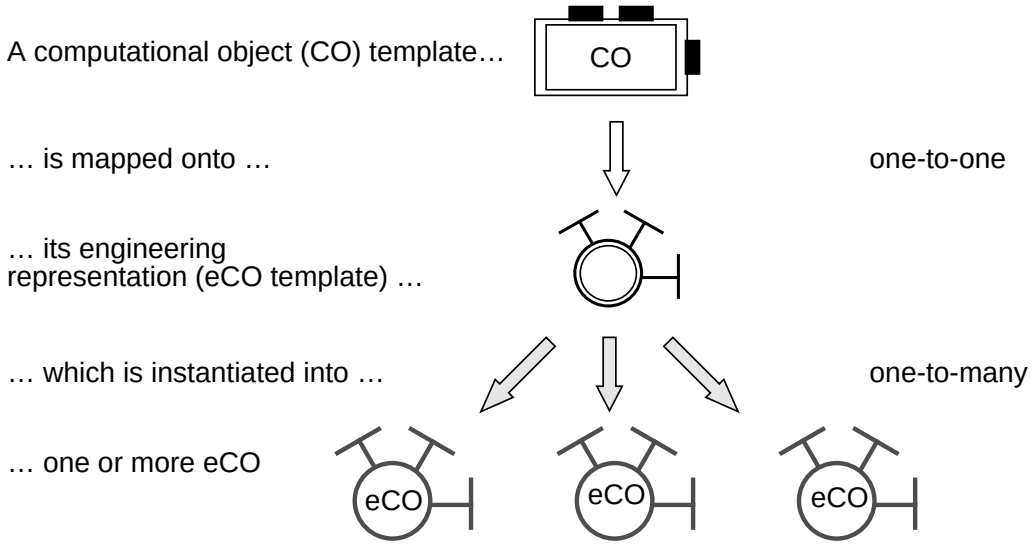


Figure 5.3: Computational Object, eCO template, and eCOs

Since the computational objects are subjected to distribution, there may exist a number of possible sets of distribution configuration. These sets of configuration are deployed under the restrictions such as the QoS requirements and the security requirements.

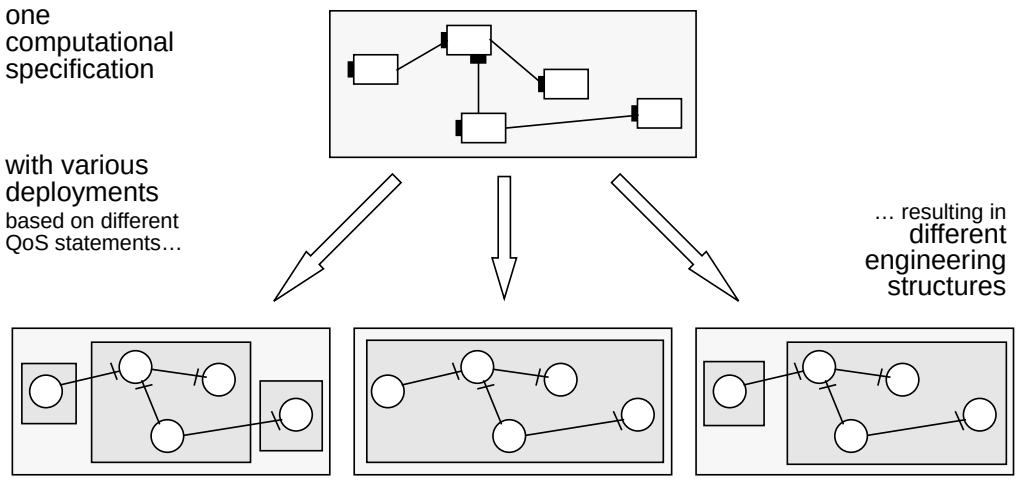


Figure 5.4: Deploying Computational Object to Engineering Object

The design of the RSMS imposes no restriction on the locations of the eCOs, except the location of the ODBC Bridge objects. Since the ODBC Bridge client does not accept CORBA connections but only local ODBC function calls, the computing node that hosts the database provider is a non-DPE node. All eCOs, which needs to contact the ODBC Bridge client, must be located in the same computing node as the ODBC Bridge client. However, different instances of ODBC Bridge client can be created to suit the needs of other eCOs. This is because all the instances of the ODBC Bridge client are able to contact the ODBC Bridge server irrespective of their location. The ODBC Bridge Server has to be located in the same computing node and the actual database engine.

The physical locations of the eCOs and the deployment strategy are discussed in chapter 6.

5.3 Implementation Code

Since the implementation codes of all the objects involved are massive, this section only discusses the highest level and most important parts of the code, including the IDL and screen outputs of the GUIs. The low level code is self-explanatory and the code is essentially the translation of the high-level design planes to the low-level language that computers can understand.

5.3.1 Basic Object Types

This section describes those basic object types but not specified in the standard TINA documentation, but defined in this system, as they are either service specific or SATINA specific components. These include the RSMS common types and the third party common types defined along the development of the RSMS system. It is important to study these basic objects types before discussing the implementation code in detail.

RSMS Common Types

The IDL file describing the RSMS common types is listed below.

```
11 module RSMSCommonTypes {
12
13        typedef TINACommonTypes::t_PropertyList t_RSMSAppSoftProperties;
14
15        struct t_RSMSAppSoft {
16                string name;
17                t_RSMSAppSoftProperties properties;
18        };
19
20        typedef sequence <t_RSMSAppSoft> t_RSMSAppSoftList;
21
22 };
```

This IDL defines a module `RSMSCommonTypes`. The module includes a TINA property list called `t_RSMSAppSoftProperties`, which is a table of undefined *name* and *property* pairs. This object type can be used to describe the properties of a particular application software such as *the hosting server ID* and *the cost*.

An object type named `t_RSMSAppSoft` is also defined to represent the application software object. It contains the `name` attribute, which is the name of the application software and the `t_RSMSAppSoftProperties` attribute, which stores the application software's properties.

A `t_RSMSAppSoftList` is defined as an array of `t_RSMSAppSoft` which is used to represent a list of application software object.

Third Party Common Types

The IDL file describing the third party common types is listed below.

```
14 module TINA3rdpartyCommonTypes {
15
16 enum t_3rdpartyUsageErrorCode {
17        UnknownUsageError,
18        ThirdPartyServerNotFound,    //
19        UsageNotAllowed,      // You don't have permission to do it
20        UsageNotAccepted,     // Owners have declined request
21        InvalidThirdPartyLogin,
22        ThridPartyServerBusy
23 };
24
25 exception e_3rdpartyUsageError {
26        t_3rdpartyUsageErrorCode errorCode;
27 };
28
29 typedef TINACommonTypes::Istring t_ThirdPtyName;
30
31 struct t_ThirdPtyInfo {
32        t_ThirdPtyName name;
33        TINACommonTypes::t_PropertyList properties;
34        TINACommonTypes::t_InterfaceList itfs;
35        TINAAccessCommonTypes::t_SessionInfo thirdpartySessionInfo;
36 };
37
38 typedef sequence<t_ThirdPtyInfo> t_ThirdPtyInfoList;
39
40 typedef TINACommonTypes::Istring t_ThirdPtyServerName;
41
42 }; //module TINA3rdpartyCommonTypes
```

The module defined in this IDL is called `TINA3rdpartyCommonTypes`. It contains the third party related error code called `t_3rdpartyUsageErrorCode`. Included in the error code are the common errors such as `UsageNotAllowed`, `UsageNotAccepted` and `InvalidThirdPartyLogin` etc. This error code is used when throwing an exception that is related to third party errors.

The most important object type defined in this IDL is the `t_ThirdPtyInfo`, which represents a connected third party information. This object type contains detailed information about the particular third party session, including the interface list of the compUSM and the object reference of the `t_SessionInfo` object.

The IDL also defines an array of the object type `t_ThirdPtyInfo`, which is used by the retailer SSM to store currently established third party sessions in *the third party sessions table*, which is discussed in the previous chapters.

5.3.2 IA, UA, PA and SF Implementation

These TINA objects are service generic and not directly related to the design of the RSMS service. Since they are standard objects defined by TINA, official documentation of the specifications of these objects can be found in [2], [4] and [1].

5.3.3 USM Implementation

There are two variations of the USM implemented in the RSMS system. One is the retailer USM (RSMSRetUSM)and the other is the third party USM (RSMS3rdUSM). These USMs inherits the interfaces of standard USM, with added service specific interfaces to support the service specific logics.

As the third party reference point is an expanded version of the retailer reference point, the functions of the retailer USM and the third party USM are almost the same. Because the RSMS does not involve multi parties (end users), the main function of the USM is to forward the client's request to the SSM. In the retailer's case, the client is the user ssUAP; while in the third party's case, the client is the retailer compUSM. The implementations of the both USMs are identical as their functions are the same.

The specifications of the standard USM interfaces can be found in [2] and [4]. There is only one service specific interface in the retailer and third party USM, named `i_PartyAppSoftManager`, which is discussed in the following section.

Table 5.1: Additional Interfaces and Client for RSMS USM

Interface	Client(s)	Event Traces	IDL
`RSMSRetUSM::i_PartyAppSoftManager`	ssUAP	fig.4.4 & A.1	B.5
`RSMS3rdUSM::i_PartyAppSoftManager`	compUSM	fig.A.1	B.8

Table 5.2: Additional Interfaces Required by RSMS USM

Server	Interfaces
RSMSRetSSM	`RSMSRetSSM::i_ProviderAppSoftmanager`
RSMS3rdSSM	`RSMS3rdSSM::i_ProviderAppSoftmanager`

`i_PartyAppSoftManager`

This interface allows the ssUAP or compUSM to request the RSMS service specific operations. It forwards the request to the SSM for processing.

The IDL of the `i_PartyAppSoftManager` is listed below.

```
14 interface i_PartyAppSoftManager {
15
16        void Request_Software_List (
17                out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
18                in TINACommonTypes::t_SessionId SessionId,
19                in TINACommonTypes::t_UserId UserID,
20                in TINACommonTypes::t_UserProperties UserProps,
21                in boolean UpdateList
22        );
23
24        void Batch_Download (
25                in RSMSCommonTypes::t_RSMSAppSoftList aList,
26                in TINACommonTypes::t_SessionId SessionId,
27                in TINACommonTypes::t_UserId UserID,
28                in TINACommonTypes::t_UserProperties UserProps
29        );
30
31        void Remote_Execute_AppSoft (
32                in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
33                in TINACommonTypes::t_SessionId SessionId,
```

```
34              in TINACommonTypes::t_UserId UserID,
35              in TINACommonTypes::t_UserProperties UserProps
36      );
37
38      void Local_Execute_AppSoft (
39              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
40              in TINACommonTypes::t_SessionId SessionId,
41              in TINACommonTypes::t_UserId UserID,
42              in TINACommonTypes::t_UserProperties UserProps
43      );
44
45 };
```

This interface provides the following operations:

- `Request_Software_List()` - allows the client to request the available software list, in the form of the return parameter `t_RSMSAppSoftList`, from the database server.
- `Batch_Download()` - allows the client to load all the software contained in the list. It is intended for future expansion use and is currently not implemented.
- `Remote_Execute_AppSoft()` - allows the client to execute the requested application software, as indicated in the input parameter `t_RSMSAppSoft`, remotely in the server.
- `Local_Execute_AppSoft()` - allows the client to run the application software locally after download. It is intended for future expansion use and is currently not implemented.

5.3.4 SSM Implementation

There are two customised SSM objects designed for the RSMS system. One is the retailer SSM (RSMSRetSSM) and the other is the third party SSM (RSMS3rdSSM). These two SSMs inherit all the interfaces provided by the standard SSM, for which the detailed explanations can be found in the official TINA documentations [2], [4] and [1]. While both of the retailer SSM and the third party SSM contain the service specific interface `i_ProviderAppSoftmanager`, the retailer SSM contains addition service generic interfaces that support third party sessions. These interfaces are `i_Provider3rdpartyReq` and `i_ProviderRegister3rdpartyInterfaces`.

Table 5.3: Additional Interfaces and Client for RSMS SSM

Interface	Client(s)	Event Traces	IDL
`RSMSRetSSM::i_ProviderAppSoftManager`	Ret USM	fig.4.4 & A.1	B.6
`RSMS3rdSSM::i_ProviderAppSoftManager`	3pty USM	fig.A.1	B.9
`TINAProvider3rdpartyUsage::` `i_Provider3rdpartyReq`	Ret SSM	fig.A.2 & A.3	B.4
`TINAProvider3rdpartyUsage::` `i_ProviderRegister3rdpartyInterfaces`	compUSM	fig.A.2	B.4

Table 5.4: Additional Interfaces Required by RSMS SSM

Server	Interfaces
Ret PA	`TINA3rdptyInitial::i_thirdPartySupport`
compUSM	`RSMSCUSM::i_CUSMAppSoftManager`
RSMSCore	`RSMSCore::i_ RSMSCoreExecManager`

`i_ProviderAppSoftManager`

This interface allows the clients (retailer USM or third part USM) to invoke the RSMS specific operations. It forwards the clients' requests to the compUSM (in case of the retailer) or RSMSCore (in case of third party) to further processing the requests.

The IDL of the `i_ProviderAppSoftManager` is very similar to `i_PartyAppSoftManager` and is listed in appendix B.6. This interface provides the operations the same as the interface `i_PartyAppSoftManager` described in section 5.3.3.

`i_Provider3rdpartyReq`

This interface is for internal use of the retailer SSM. It allows the other interfaces, such as the `i_ProviderAppSoftManager`, in the retailer SSM to start or end a third party session.

The IDL of the `i_Provider3rdpartyReq` is listed below.

```
40 interface i_Provider3rdpartyReq {
41
```

```
42        void Start_3pty_Service (
43                in TINACommonTypes::t_SessionId SessionId,
44                in TINACommonTypes::t_UserId UserID,
45                in TINAProviderAccess::t_ApplicationInfo app,
46                in TINAAccessCommonTypes::t_ServiceId serviceId,
47                in TINACommonTypes::t_UserId thirdptyLogin,
48                in TINACommonTypes::t_UserProperties thridptyProps
49        ) raises (
50                TINAUsageCommonTypes::e_UsageError,
51                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
52        );
53
54        void End_3pty_Service (
55                in TINACommonTypes::t_SessionId SessionId,
56                in TINACommonTypes::t_ParticipantSecretId SecretId,
57                in TINA3rdpartyCommonTypes::t_ThirdPtyServerName ServerName
58        ) raises (
59                TINAUsageCommonTypes::e_UsageError,
60                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
61        );
```

This interface provides the following operations:

- `Start_3pty_Service()` - allows the other interfaces in the SSM to request for starting a third party session. The authentication information, such as the third party ID, the login and password to the third party server and the service ID of the third party service should be provided as the input parameter.
- `End_3pty_Service ()` - allows the other interfaces in the SSM to end an established third party session by provider the third party server ID and the session ID.

`i_ProviderRegister3rdpartyInterfaces`

This interface only contains one operation called `registerInterfaces()`. It provides the function for the newly created compUSM objects to register their interfaces to the retailer SSM.

The IDL of the `i_ProviderRegister3rdpartyInterfaces` is listed below.

```
27 interface i_ProviderRegister3rdpartyInterfaces {
28
29         void registerInterfaces (
30                 in TINACommonTypes::t_ParticipantSecretId myId,
31                 inout TINACommonTypes::t_RegisterInterfaceList itfs,
32                 in TINA3rdpartyCommonTypes::t_ThirdPtyServerName ServerName
33         ) raises (
34                 TINAUsageCommonTypes::e_UsageError,
35                 TINACommonTypes::e_InterfacesError,
36                 TINACommonTypes::e_RegisterError
37         );
38 };      // interface i_Provider3rdpartyInterfaces
```

5.3.5 CompUSM Implementation

The full specification of the compUSM is not defined in any of the official TINA documentations. The only information given is that the compUSM, similar to the USM, is inherited from the Member User Session Manager object (MUSM)[1]. Thus, the normal USM is applied as a template when implementing the compUSM object. A service specific compUSM (RSMSCUSM) is implemented with two interfaces, `i_CUSMAppSoftManager` and `i_ProviderBasicReq`. The former is a service specific interface while the later is a service generic interface.

A CompUSMs is instantiated when a new third party service session is created. Therefore, each compUSM instance is associated with a particular instance of a third party service session. The compUSM is essentially a proxy between the retailer and its corresponding third party service provider.

Table 5.5: Interfaces and Client for compUSM

Interface	Client(s)	Event Traces	IDL
`RSMSCUSM::i_CUSMAppSoftManager`	Ret SSM	fig.A.1	B.7
`TINAProviderBasicUsage::` `i_ProviderBasicReq`	Ret SSM	fig.A.3	ref to [2]

Table 5.6: compUSM Required Interfaces

Server	Interfaces
3pty USM	`RSMS3rdUSM::i_PartyAppSoftManager`
Ret SSM	`TINAProvider3rdpartyUsage::` `i_ProviderRegister3rdpartyInterfaces`

`i_CUSMAppSoftManager`

This interface allows the retailer SSM to request for the RSMS specific operations. It forwards the request to its corresponding third party's USM to further processing.

The IDL of the `i_CUSMAppSoftManager` is very similar to `i_PartyAppSoftManager` and is listed in appendix B.7.

As the compUSM is the proxy between the third party and the retailer, the operations provided by this interface is the same as the interface `i_PartyAppSoftManager` described in section 5.3.3.

`i_ProviderBasicReq`

This interface is a standard TINA interface which is revealed in [2]. It supports many operations but only one is relevant to the RSMS service, which is the `endSessionReq()`. This operation is implemented to terminate the service session and to free up the system resources used by the compUSM object.

5.3.6 Retailer PA Implementation

Since there is lack of definition of the third party reference point from TINA Consortium, this system uses a cascaded retailer reference point for third party access. To use the retailer reference points to access the third party provider, there should be a Provider Agent (PA) in the retailer domain, the same as there is a PA in the consumer domain. Thus, the retailer PA is created to accomplish this requirement. The retailer PAs are used to represent third party providers in the retailer domain. It is designed to be service generic, which means that other services can use this PA without any customisation or modification.

The retailer PA is developed base on the standard PA implementation, which can be found in [2]. In addition to the standard PA interfaces, the retailer PA has the

interface `i_thirdPartySupport` for accessing the third party domain.

Table 5.7: Additional Interfaces and Client for Retailer PA

Interface	Client(s)	Event Traces	IDL
`TINA3rdptyInitial::i_thirdPartySupport`	Ret SSM	fig.A.2 & A.3	B.3

Table 5.8: Additional Interfaces Required by Retailer PA

Server	Interfaces
None	None

`i_thirdPartySupport`

This interface allows the retailer SSM to contact the third party that the retailer PA represents. The retailer SSM can start the third party service session on demand when needed. It takes the authentication information from the retailer SSM and login to the third party domain using the same procedure as the consumer PA login to the retailer domain.

The IDL of the `i_thirdPartySupport` is listed below.

```
17 interface i_thirdPartySupport {
18
19        void Start_3pty_Service (
20                out TINAAccessCommonTypes::t_SessionInfo sessionInfo,
21                in TINAAccessCommonTypes::t_ServiceId serviceId,
22                in TINACommonTypes::t_UserId UserID,
23                in TINACommonTypes::t_UserProperties UserProps,
24                in TINAProviderAccess::t_ApplicationInfo app
25        ) raises (
26                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
27        );
28
29        void End_3pty_Service (
30                in TINAAccessCommonTypes::t_SessionInfo sessionInfo
31        ) raises (
32                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
33        );
34
```

```
35 };
```

This interface provides the following operations:

- `Start_3pty_Service()` - allows the retailer SSM to establish the access session and service session to the third party service provider.
- `End_3pty_Service()` - allows the retailer SSM to terminate the third party access session and service session.

5.3.7 RSMSCore Implementation

The RSMSCore object is the central component of the RSMS service. It contains all the intelligence to render the ASP service, contrary to the other objects that only contain architectural supporting functions. The RSMSCore object a Service Support Component (SSC) defined in [1]. It contains one stream interface which is used to transfer both the screen output data and user input data, between the third party content provider and the consumer. Three interfaces are included in this object, namely `i_RSMSCoreExecManager`, `i_RSMSCoreDownloadManager` and `i_ProviderBasicReq`.

Table 5.9: Interfaces and Client for RSMSCore

Interface	Client(s)	Event Traces	IDL
`RSMSCore::i_RSMSCoreExecManager`	3pty SSM	fig.4.4	B.10
`RSMSCore::i_RSMSCoreDownloadManager`	3pty SSM	None	None
`TINAProviderBasicUsage::i_ProviderBasicReq`	3pty SSM	fig.A.3	ref. to [2]

Table 5.10: RSMSCore Required Interfaces

Server	Interfaces
3pty SSM	`TINAProviderBasicUsage::` `i_ProviderRegisterInterfaces`

`i_RSMSCoreExecManager`

This interface allows the third party SSM to forward the remote execution requests originated from the consumer. This interface will create a new process in the operating system to execute the application software after it has received a remote

execution request.

The IDL of the `i_RSMSCoreExecManager` is listed below.

```
35 interface i_RSMSCoreExecManager {
36
37        void Request_Software_List (
38                out RSMSCommonTypes::t_RSMSAppSoftList RequestedList,
39                in TINACommonTypes::t_SessionId SessionId,
40                in TINACommonTypes::t_UserId UserID,
41                in TINACommonTypes::t_UserProperties UserProps,
42                in boolean UpdateList
43        );
44
45        void Remote_Execute_AppSoft (
46                in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
47                in TINACommonTypes::t_SessionId SessionId,
48                in TINACommonTypes::t_UserId UserID,
49                in TINACommonTypes::t_UserProperties UserProps
50        );
51 };
```

This interface provides the following operations:

- `Request_Software_List()` - allows the client to retrieve the available software list stored in the central database. The current implementation relies on the third party content providers to export their available software lists to the central database. However this is operation can be implemented in such a way that the retailer actively polls the available software list on each of the contracted third party providers, without modification of the existing interfaces. This operation places the available software list to the output parameter `RequestList` when the function returns.

- `Remote_Execute_AppSoft()` - executes the application software and exports the display to the consumer through an established TCP/IP stream. The input parameter `_AppSoft` indicates which application to execute. The other input parameter `UserProps` contains user properties such as the IP address of the user's terminal, which is used when exporting the display.

`i_RSMSCoreDownloadManager`

This interface allows the third party SSM to forward the batch-download requests from the retailer side, for remote software installation and management purpose. Although the IDL of this interface is defined, it is intended for future expansion use and it is currently not implemented.

The IDL of the `i_RSMSCoreDownloadManager` is listed below.

```
17 interface i_RSMSCoreDownloadManager {
18
19        void Request_Software_List (
20                out RSMSCommonTypes::t_RSMSAppSoftList RequestedList,
21                in TINACommonTypes::t_SessionId SessionId,
22                in TINACommonTypes::t_UserId UserID,
23                in TINACommonTypes::t_UserProperties UserProps,
24                in boolean UpdateList
25        );
26
27        void Batch_Download (
28                in RSMSCommonTypes::t_RSMSAppSoftList aList,
29                in TINACommonTypes::t_SessionId SessionId,
30                in TINACommonTypes::t_UserId UserID,
31                in TINACommonTypes::t_UserProperties UserProps
32        );
33 };
```

This interface provides the following operations:

- `Request_Software_List()` - It has the same functionality as the one defined in the `i_RSMSCoreExecManager` interface. The reason for duplicating this operation is that the third party content provider can choose to support either the interface `i_RSMSCoreExecManager` or the interface `i_RSMSCoreDownloadManager`, and still has the `Request_Software_List()` function included.
- `Batch_Download()` - Uploads one or more requested software packages to the user's terminal and performs installation remotely.

`i_ProviderBasicReq`

This interface is the same as the `i_ProviderBasicReq` described in section 5.3.5.

5.3.8 ssUAP Implementation

To accommodate different computing platforms of the user terminal, three different favours of the *service specific user application* are implemented. The first one runs on the Linux platform, the second one runs on the MS Windows platform and the third one runs on the JAVA virtual machine. The Linux and Windows version have built-in PA while the JAVA version relies on the SATINA service generic PA, which is also written in JAVA. The JAVA ssUAP is implemented as a class that the service generic PA can be call upon when needed.

The Linux and Windows version are designed to integrate seamlessly with the service generic PA. Once the user has created the access session through the JAVA PA, the JAVA PA initiates the native Linux or Windows ssUAP, with the access references passed as the command-line parameter. The Linux or Windows ssUAP then uses the existing access session to create the RSMS service session. If the access reference is invalid or nothing is passed from the command-line, Linux or Windows ssUAP will use its own build-in PA to prompt for the user login and password to create a new access session.

Currently the ssUAP is implemented as a pure client object, i.e. there is no operational interface for the other object to connect to it. However, it contains one stream-binding interface.

Table 5.11: Interfaces and Client for RSMS ssUAP

Interface	Client(s)	Event Traces	IDL
None	None	None	None

Table 5.12: RSMS ssUAP Required Interfaces

Server	Interfaces
Ret USM	`RSMSRetUSM::i_PartyAppSoftManager`

Graphical User Interfaces Implementation

This section describes the typical sequence of the appearances of the graphic user interfaces provided by the ssUAP. These interfaces may look slight different from the Linux, Windows and JAVA implementations due to different GUI components used[1]. However, the operation methods are the same among the three.

1. Login Dialog

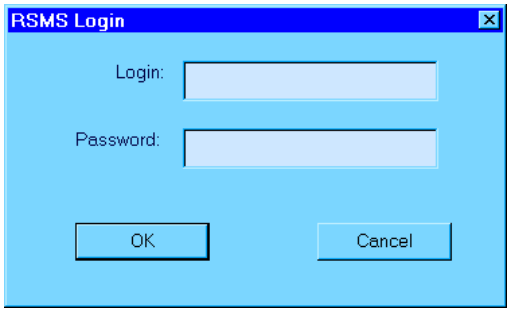


Figure 5.5: Login Dialog

If the SATINA service generic PA is not used to initiate the ssUAP, i.e. the access reference is not provided as a command-line parameter when the ssUAP starts, the login dialog will appear. This prompts the user to enter the login ID and password, in order for the build-in PA to create a new access session. This dialog will not be displayed if the access reference passed form the service generic PA is pointing to a valid access session. The GUI is shown in figure 5.5.

2. Software List Dialog

Once the ssUAP has received the valid reference to access session, it will start the RSMS service session and automatically request for a list of available software, using the operation `Request_Software_List()`, on the interface `i_PartyAppSoftManager` of the retailer USM. The returned list is then displayed in a dialog similar to one shown in figure 5.6. The titles of the application software are displayed on the list-box in the middle of the dialog. Clicking the "Refresh" button causes the ssUAP to re-request for the available software list from the server. If the user double clicks on a software title on the list or if the user has selected one title by clicking the title once and then clicks the "OK" button, the *Software Details Dialog* would appear. If the "Cancel" button is clicked, the dialog would be closed and would cause the

[1]The screen captures shown in this section are taken from the Windows version of the ssUAP.

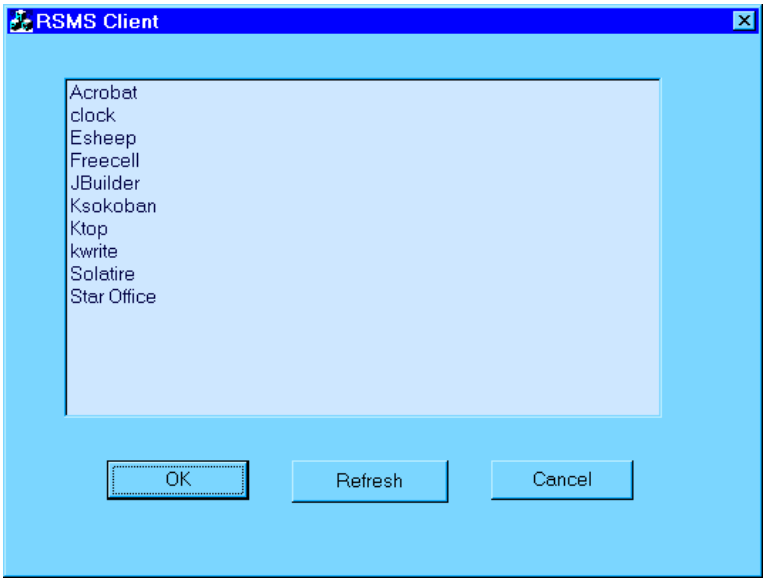


Figure 5.6: Software List Dialog

ssUAP to terminate the access session, using the operation `endSessionReq()` on the interfaces `i_ProviderBasicReq` on the retailer USM.

3. Software Details Dialog

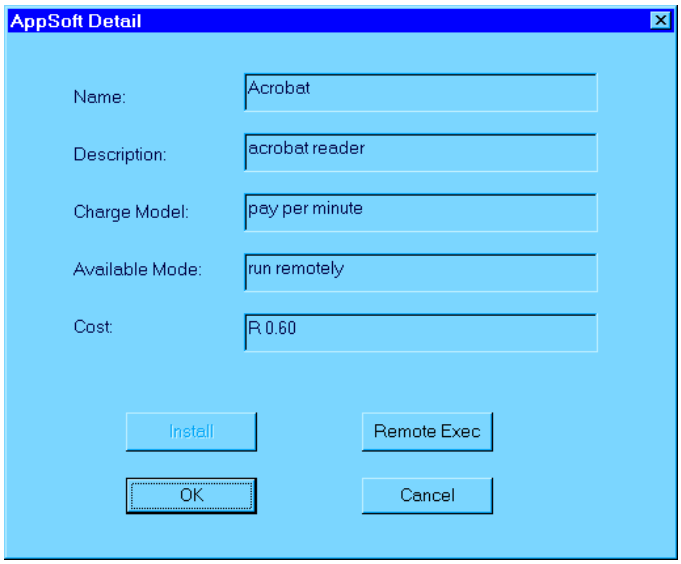


Figure 5.7: Software Details Dialog

This dialog displays the details of the application software selected by the user, which is shown in figure 5.7. The ssUAP extracts the `AppSoft` object of the selected software from the array `AppSoftList`. From the attribute `t_RSMSAppSoftProperties` of the extracted `AppSoft` object[2], the ssUAP can

[2]See section 5.3.1

determine all the information that the user needs to know about the particular application software.

If the "Remote Execute" button is clicked, the operation `Remote_Execute_AppSoft()`, on the interface `i_PartyAppSoftManager` will be invoked on the retailer USM. If the execution of the application software is successful, the screen output of the software will be displayed on the user's terminal.

If the "Install" button is clicked, the operation `Batch_Download()`, which suppose to remotely install and manage of the application software, will be called on the retailer USM. Currently this feature is disabled.

Either the "OK" or the "Cancel" button will cause this dialog to close and return to the Software List Dialog.

5.3.9 ODBC Bridges and Database Engine Implementation

The ODBC Bridge Client and Server are implemented using commercially available software package. The ODBC Bridge Client and Server does not run on top of the TINA DPE, rather they are at the same level of the TINA DPE, which is running directly on top of the native computing environment of a computing node. The ODBC Bridge Client has to register its existence on its local system, typically by assigning a Data Source Name (DSN) through the ODBC administration interface offered by the operating system. The ODBC Bridge is running as a daemon on top of the native computing environment and listening to the requests from the ODBC Bridge Client on a specific TCP port. The ODBC Bridge Client has to be configured so that it contains the IP address and the port number of the ODBC Bridge Server.

In Microsoft Windows environment, ODBC is a build-in feature. On the other hand, in the Unix environment, the freely available Unix-ODBC package has to be installed. ODBC is a database middleware that isolates the database client from the underlying database engine. To use the database engine through the ODBC interface, the database engine vendor has to provide a specific ODBC driver for the particular database engine. The ODBC then uses this drive to access the database and provides standardise access functions to the client applications.

The components that need access to the central database, e.g. the retailer SSM or the third party SSM, have to invoke the standard ODBC function call on the local system, using the DSN that registered by the ODBC Bridge Client. The ODBC Bridge Client uses the pre-determined address to forward the database call to the ODBC Bridge Server. The ODBC Bridge Server then access the database engine

through the local ODBC function call.

Database Structure Implementation

The actual database engine implemented in the RSMS system uses relational database to store the database entries. Structured Query Language (SQL) is used to query or modify the database.

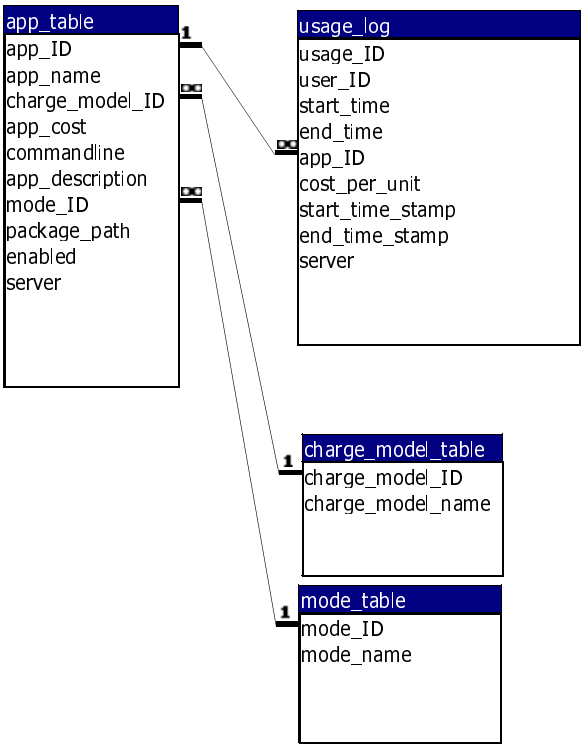


Figure 5.8: Relational Database Tables

The database structure is shown in figure 5.8. It contains four tables namely *app_table*, *usage_log*, *charge_model_table* and *mode_table*, with primary keys *app_ID*, *usage_ID*, *charge_model_ID* and *mode_ID* respectively. These primary keys are used to index and cross referencing between the tables.

The *mode_table* contains a table of the names of the service modes. Example of the service modes are "run remotely" and "install locally". Currently only the "run remotely" mode is supported.

The *charge_model_table* contains a table of charging models. These models are supposed to be used by the service generic billing service. Examples of the charge models are "pay per use" and "pay per minute".

The *app_table* contains a table of available application software supported by all the third party service providers. It contains detailed information about the application software, such as the application name and the third party server ID. It also includes *charge_model_ID* and *mode_ID* of the application software, which can be used as references to the table *charge_model_table* and *mode_table*.

Each entry in the *usage_log* contains information about a particular usage event of the third party content provider. Information such as *start_time*, *end_time* and *cost* are recorded. Each usage entry also contains *app_ID* in such a way that the details of the application used can be lookup in the *app_table*.

5.3.10 Retailer and 3rd party Admin GUI Implementation

The retailer and third party administration GUIs are implemented as two normal applications, which only invoke the local ODBC calls without any CORBA capability. Therefore, they can only communicate with the ODBC Bridge Client but not the other TINA objects. These two admin GUIs can be located in any of the computing node within the system as long as there is an instance of the ODBC Bridge Client running on the same node. They access the database directly through the ODBC Bridges similar to those procedures discussed in section 5.3.9.

The main difference between the retailer admin GUI and the 3pty admin GUI is that the retailer admin GUI can access the entire content of the database while the 3pty admin GUI can only access the content that is related to its server. This restriction is implemented by forcing the 3pty admin GUI to include some specific SQL search criteria.

Graphical User Interfaces Implementation

This section describes various dialogs implemented for the retailer admin GUI and the 3pty admin GUI.

- Admin GUI Main Menu

  Figure 5.9 shows the main menu of the retailer admin GUI. It contains two groups of buttons namely “List Operations” and “Register”. The visual difference between the retailer and third party admin GUI is that both the “Modify Modes” button and “Modify Charge Models” button do not exist in the main menu of the third party admin GUI.

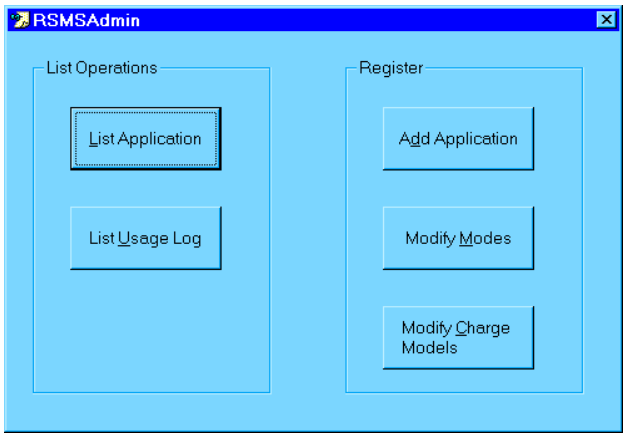


Figure 5.9: Ret Admin GUI Main Menu

When the user clicks the "Add Application" button, the *Add Application Dialog* will appear. When the user clicks the "List Application" button, the *Application List Dialog* will be displayed. If the "List Usage Log" button is clicked, the *Usage Records List Dialog* will appear. These dialogs are discussed in the subsequence sections.

Clicking the "Modify Modes" button, a dialog will appear which allows the user to change the entries in the table *mode_table* described in section 5.3.9. Clicking the "Modify Charge Models" button opens a dialog that enables the user to modify the entries in the table *charge_model_table*. Since the two dialogs are very simple, there is no need to discuss them further in this section.

- Add Application Dialog

  This dialog is almost the same for the retailer admin GUI and the third party admin GUI except that the "Server Name" field in the third party admin GUI is fixed to the third party content provider's ID. This is to ensure that the third party administers can only create new entries for their own server. This dialog enables the retailer or the third party to register the available software to the central database by creating database entries in the *appt_table* discussed in the section 5.3.9.

  This dialog requires the administrators to enter the details of one application software. Some of the fields are optional and others are compulsory. Clicking the "update" button will cause the GUI to check if all the compulsory fields have been entered. The system prompts the administrator to enter all the compulsory fields if any one of them is empty. Otherwise, the system will send an SQL to the central database, through the ODBC Bridges, to create a new entry in the *app_table*.

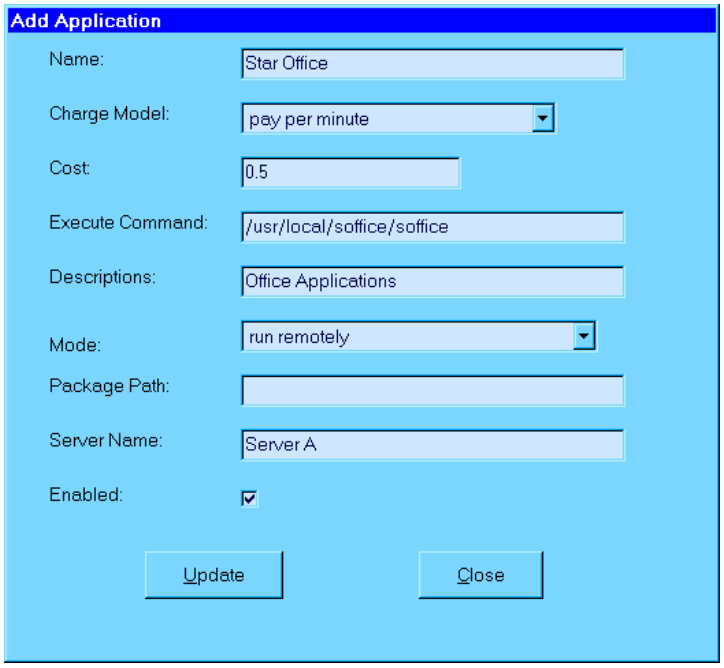


Figure 5.10: Add Application Dialog

There is a special field called "Enable", which is a Boolean field. If the value of the field is FALSE, the entry will not be included in the returned application software list when the user's ssUAP invokes the `Request_Software_List()` operation on the retailer USM, i.e. it is not visible by the user.

- Application List Dialog

  Figure 5.11 shows the Application List Dialog. It displays the contents of the table *app_table* in a list, according to the search criteria given by the administrator. There are three main parts in the dialog: the top part of the dialog contains four textboxes, which are for the administrator to enter the search criteria. The middle part of the dialog enables the administrator to decide on how to sort the list. The bottom part is a listbox showing the result of the search.

  The system will formulate a SQL statement according to the search criteria, once the "Query" button is clicked. The formulated SQL statement is then sent to the database provider. Once the result has been fetched, the listbox will be refreshed. The third party admin GUI does not contain the search criteria "Server Name". For the third party admin GUI, the list only shows the software entries that belong to the third party that the GUI is designed for. The third party ID is customised during the installation process of the third party admin GUI and cannot be changed afterward.

  Double clicking a particular entry on the listbox will cause a new dialog, which

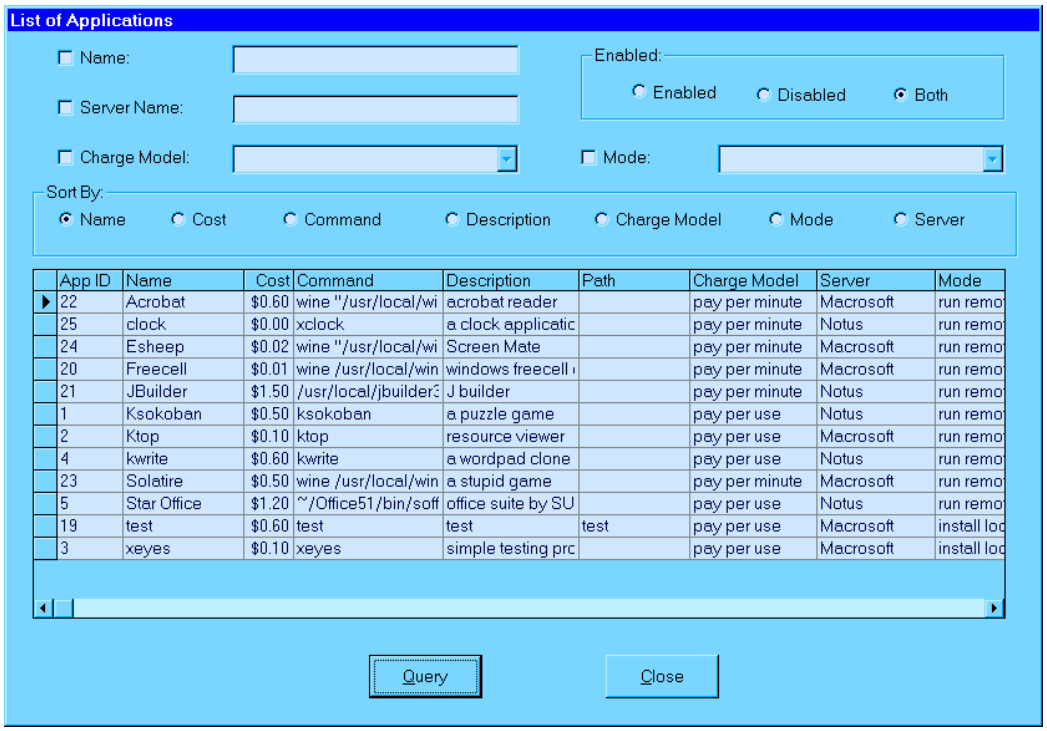


Figure 5.11: Application List Dialog

looks very similar to the Add Application Dialog, to appear. The administrators can use this dialog to modify the information of the chosen application software entry in the database.

- Usage Records List Dialog

  Similar to the Application List Dialog, the Usage Records List Dialog, which is shown in figure 5.12, is partitioned into three parts. The first part contains five textboxes for the administrator to enter the list criteria. The second part allows the administrator to sort the list by a specific field. The third part is a listbox showing the search result. The dialog in the third party admin GUI, as expected, does not have the field "Server Name". For the third party admin GUI, the search result listed in the bottom part of the dialog shows only the records that are related to the third party.

  The listbox shows the usage record entries in the table *usage_log* in the database according to the criteria specified. The list includes information such as the starting time, ending time, user ID, the application name and the third party server ID etc. The total cost of all the entries is calculated and displayed on the dialog.

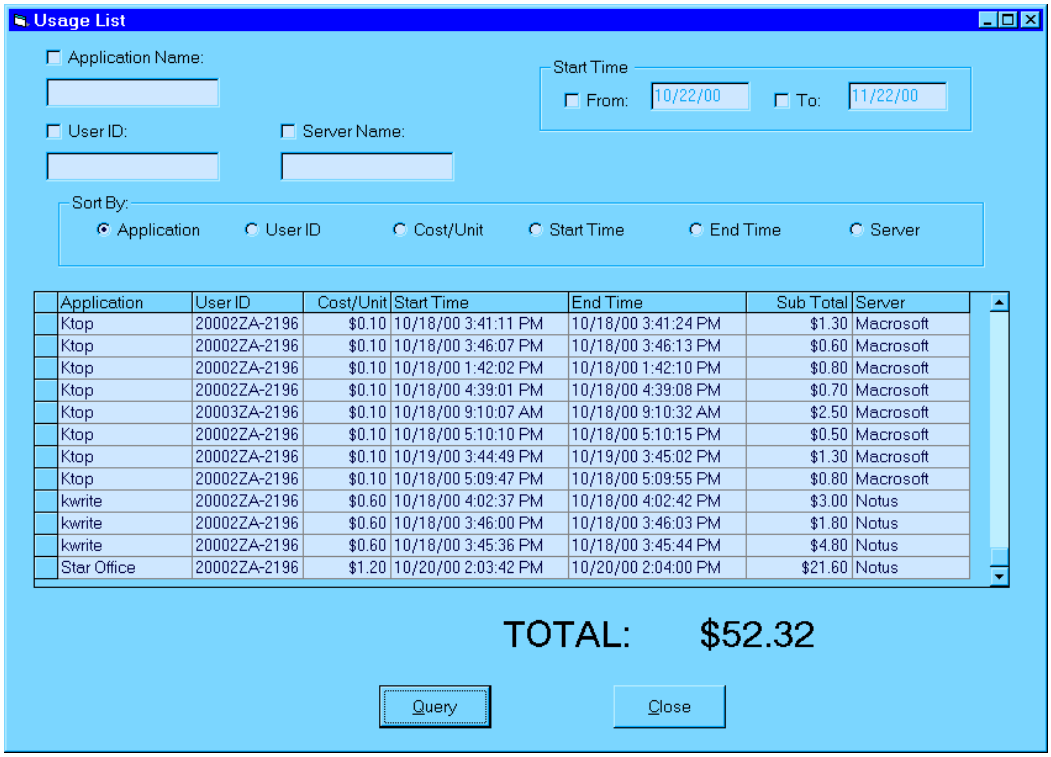


Figure 5.12: Usage Records List Dialog

5.3.11 Web Interface Implementation

A web interface is provided for the users to check their usage, with the convenience of using a standard browser. A Visual Basic (VB) script is written on the web page to contact the database provider through the ODBC Bridges. After the user has entered the user ID, the web server executes the VB script and formulates a SQL statement. The web server sends the SQL query to the ODBC Bridge Client, through the local ODBC call. The SQL statement is forward all the way to the database engine and the query is then actually carried out. The query result is passed back to the web server, which converts the result to a HTML page. The page is then sent back to the browser. The result page appears to the client as a normal web page and the execution of the script is transparent to the client. Figure 5.13 shows the snapshot of a standard browser displaying an usage record.

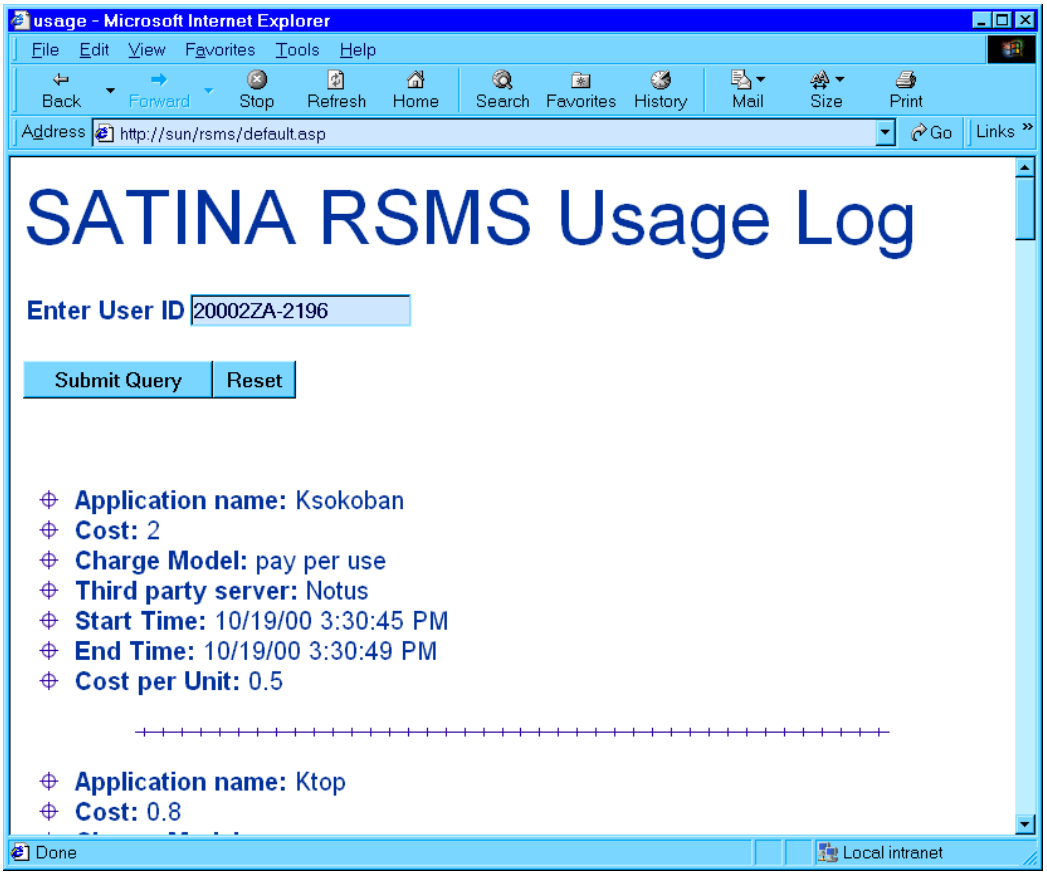


Figure 5.13: Web Interface for Usage Record Queries

Chapter 6

Service Implementation: Physical Plane

6.1 Introduction

The last plane of the service development model is the physical plane. It is concerned with the operational deployment and execution of the software modules. The outputs of this plane are the *technology model* and the *physical deployment model*. A technology model describes in detail the software technology, hardware technology, and the topology of the system. The physical deployment model shows the location of the software modules in terms of the computing nodes.

6.2 Technology Model

The technology model is one of the outputs of the physical plane. It corresponds to the technology viewpoint of the RM-ODP standard. This viewpoint focuses on the choice of technology to support the system. To describe the choice of technology, the technology model is separated into two parts, the *hardware technology* and the *software technology*.

6.2.1 Hardware Technology

To mimic the typical scenario of the ASP service, six workstations in the SATINA platform were used. The names of these machines are: *Fire*, *Ruby*, *Sun*, *Mint*,

Glacier and *Sky*. All these machines are X86-processor based and have standard components installed, such as the hard disk and the CD-Rom driver. In addition to the standard components, all the machines have one or more network interface cards (NIC) equipped. All these NIC support 100Mpbs Fast Ethernet connections, which provide a common protocol for the machines to communicate with each other. Two Ethernet switches are used to have all these six machines connected[1].

No special hardware technology is required to support the RSMS system. However, there are a few requirements on the hardware configurations, which are listed below.

- All the machine should have some means of connection to the rest of the machine. Currently only Ethernet is used.
- The hard disk should have enough capacity to store all the assigned software components or modules.
- The processor and memory space must be capable of running the assigned tasks. For example the third party content provider must be able to run at least a few remote applications simultaneously.
- The bandwidth is wide enough that the delay experienced by the user is acceptable[2].

6.2.2 Software Technology

This section discusses various software technologies employed on the system to support the RSMS service.

Operating System

To create a heterogeneous computing environment, three computers (Fire, Ruby and Mint) have Linux (RedHat 6.x distributions) operating system installed. The rest of the computers (Glacier, Sun and Sky) have Microsoft Window NT 4 (with service

[1] refer to section 6.4

[2] The average user activities (mouse and keyboard) use about 3-4kbytes per second to stream between the server and the client terminal. Screen changes on the remote application cause bursts in the network. The time to display screen changes depends on the area of change, screen resolution, bandwidth between the user terminal and the server, and the processing speed of the user terminal and the server.

pack 3+) operating system installed. On top of the OS, the TCP/IP network stake is installed on all of the machines.

Distributed Processing Environment

The open source Mico Object Request Broker (ORB) is used to provide the basic functions of a standard TINA DPE for all the programs written in C++ (Both Windows and Linux platform). JAVA applications use Orbacus ORB to provide basic DPE functions. These two ORBs are compatible with each other, therefore the components sitting on top of different ORBs are able to communicate with each other.

IA, UA, SF, USM, SSM, CompUSM, RSMSCore and Retailer PA

The IA, UA, SF, USM, SSM, CompUSM, RSMSCore and Retailer PA objects are all implemented in C++ and compiled using GNU C Compiler (GCC). The implementation code of these components has not been ported in the Windows environment. Therefore, these components have to be deployed on the nodes that have Linux installed as the operating system.

For implementation purpose, the engineering objects (e.g. IA and UA) are mapped into executable binaries; the interfaces (e.g. `i_PartyAppSoftManager`) of these engineering objects are mapped into C++ classes; the operations (e.g. `Request_Software_List()`) on the interfaces are mapped into C++ methods (or functions). Therefore, creating an instance of an USM object is equivalent to running the USM executable on the system. Similarly, passing the reference of an interface to another engineering objects is equivalent to passing a pointer of an instance of an object class in C++. Invoking an operation on an interface of an engineering object is the same as calling a method on a C++ object.

PA

The PA is written in JAVA version 1.2. Due to incompatible reasons, the build-in Visibroker ORB on JAVA 1.2 platform is overrode by the use of Orbacus ORB. This is a service generic component and is able to call the JAVA class of the ssUAP when needed. Since it is written for JAVA virtual machine, the same compiled binary can be executed under different operating system.

Different ssUAPs have been built in this project: an ASCII text based ssUAP for both Linux and Windows operating system; a Microsoft Foundation Classes (MFC) based ssUAP for Windows OS; a QT[3] based ssUAP for Linux OS; and a Java based ssUAP. The compiler used under Linux system is the GNU C compiler and the compiler used under Windows is the Visual C++ compiler.

All the clients developed, except the JAVA version, have build-in PA component which can be used as standalone application. The JAVA version has to use the service generic PA. The build-in PA enables the ssUAP to create their own access sessions to request for services or if the access IOR is passed as the command-line argument, they can use the existing access sessions. This allows the C++ compiled ssUAP to run independently of the JAVA components (service generic PA). Thus, any computing node without the JAVA virtual machine installed can still be able to run the C++ version of the ssUAP.

For the user's terminal to display the screen output of the application software running in the remote server, a X-server is required to be installed in the user's terminal. The X-server uses X-protocol to send all the events such as the mouse clicks and keyboard scan-codes to the X-client. The X-client is used by the hosted applications that are running in the remote server. The X-server is used to interpret the screen output data sent from the X-client and redraw the screen on the monitor of the user's terminal. Therefore, the X-server is regarded as a part of the ssUAP.

For the nodes that are running under Linux OS, Xfree86 X-server is used. This X-server is a standard feature that comes with most of the Linux distributions. For the nodes that are running under Windows, an evaluation version of the X-server from Starnet Corporation is used. This X-server can use the native Microsoft Windows as the windows manager of the Unix system. This feature enables the remote application running in the remote server has the look and feel as if it is running locally.

The X-server can be regard as a dumb terminal with graphics and mouse functions. The reasons for the popularity of the dumb terminal model in the past are: firstly, the costs of the dumb terminal are low. Secondly, they require little bandwidth. Thirdly, the user interface (ASCII characters) is standard. X-server has similar characteristics compared to the traditional dumb terminal. It costs little and it is standard in the Unix community. Furthermore, network operators are continuously

[3] A portable, generic GUI library

upgrading their networks to support multimedia applications, which results in the bandwidth used by X-protocol appears to be relatively small compared with audio or video streaming. Since the graphical interface is a necessity nowadays, thin clients equipped with X-server or a standard browser will probably replace the traditional, dumb and text based terminals in the future.

Naming Service

The naming service provided by the Mico Orb is used as the simplified broker function in the system. It is compiled as a standalone executable and can be run in background as an independent process. It is written in C++ and is portable to Windows or other Unix platform. The naming service allows the third party content providers to register their PA's object references.

Database Provider

The commercially available software package from EasySoft Corporation is used for the ODBC Bridge Server and ODBC Bridge Client components. As the ODBC Bridge does not accept CORBA connections, it does not run on top of the TINA DPE. Therefore, the ODBC Bridge components can be located on non-DPE nodes. Although they can also be deployed on DPE node, co-existing with other engineering object, they do not communicate to the other objects through the DPE.

In order to contact the ODBC Bridge Client, the contacting party should invoke the ODBC function call on the local operating system. The ODBC functionality is build-in for Microsoft Windows system. For Unix systems, the open source package Unix-ODBC is used to provide the local ODBC interface.

The actual database used is the Microsoft Jet Engine. The database tables discussed in section 5.3.9 in Chapter 5 is implemented using Microsoft Access. As the ODBC Bridge Server needs to contact the database driver locally to actually issue the SQL commands to the database, the Microsoft Access ODBC driver has to be installed on the machine that hosts the database.

The advantages of using ODBC are its simplicity and its portability. Database engines produced by different software vender can be used, as long as a supported database driver is installed, i.e. the database clients do not depend on a particular database implementation. For example, the database used in this implementation

is Microsoft Access, but it can be changed to other database engine without modification of the existing objects. Combined with the ODBC bridge, the database can be located anywhere in the network, i.e. the database is location transparent. In addition, many different programming languages, such as C, Visual Basic and JAVA (through JDBC), support the ODBC functions.

Administration GUIs

The administration GUIs (Retailer Admin GUI and Third party Admin GUI) are developed using Microsoft Visual Basic. The reason for using Visual Basic is that it is simple and fast to build.

To access the service data, the administration GUIs communicate with the ODBC Bridge Client directly through the local ODBC calls. In addition, the GUIs do not need to contact with the other engineering objects. Therefore, these components can be located on the non-DPE nodes.

Web Server

Microsoft Internet Information Server is used as the web server. The RSMS web interface for the usage records is written in Visual Basic (VB) script, which is stored in an active server page. The web page relies on the web server to carry out the VB script to communicate with the database provider.

6.3 Physical Deployment Model

To simulate the multi-service providers environment, a RSMS service provider (retailer), a database provider, two content providers (third party), a naming service provider and two users are produced. The service components of these stakeholders are spread across the six computers[4]. Table 6.1 shows the distributions of the components on the six computing nodes, together with their roles and, the operating system used and the orb used.

The RSMS service provider (retailer) occupies two computing node, Fire and Sun.

[4]There is not necessary to have one-to-one relationship between the computing node and the service provider. Many service providers can share a single computing node while a service provider may possess many different nodes.

Table 6.1: Components Deployment Table

	Fire	Ruby	Sun	Sky	Glacier	Mint
OS	Linux 2.2	Linux 2.2	Windows NT 4	Windows NT 4	Windows NT 4	Linux 2.2
Role(s)	Retailer	Content Provider *A*	Database Provider	Retailer Admin	Consumer I	Consumer II
	Broker	Content Provider *B*		3pty Admin *A*		
				3pty Admin *B*		
ORB	Mico	Mico			Mico	Mico
					Orbacus	Orbacus
Components	IA	IA (*A*,*B*)	Access ODBC Driver	Retailer Admin GUI	PA	PA
	UA	UA (*A*,*B*)	Database Engine	3pty Admin GUI (*A*)	ssUAP (JAVA)	ssUAP (JAVA)
	SF	SF (*A*,*B*)		3pty Admin GUI (*B*)	ssUAP (MFC)	ssUAP (Qt)
	Ret PA (*A*,*B*)	App Software Binaries (X-clients)			X-Win32 (X-server)	XFree86 (X-server)
	RSMSRetUSM	RSMS3rdUSM (*A*,*B*)	Web Server		Web Browser	Web Browser
	RSMSRetSSM	RSMS3rdSSM (*A*,*B*)				
	RSMSCompUSM	RSMSCore (*A*,*B*)				
	ODBC Bridge Client	ODBC Bridge Client	ODBC Bridge Client	ODBC Bridge Client		
	Naming Service	Windows Emulator	ODBC Bridge Server			

All the retailer components (IA, UA, SF, RSMSRetUSM, RSMSRetSSM, RSMSCompUSM) are deployed on Fire. In addition to the retailer components, Fire also contains some components that belong to the naming service provider and the third party content providers (Ret PAs). There are two instances of Ret PA, which represent the two content providers. The retailer admin GUI is not located on fire but it is located on the node with the other admin GUIs, Sky.

There are two content providers, A and B. These two content providers are running separately and independently on the same node Ruby. Therefore, there are two instances of the components IA, UA, SF, etc., one for each content provider. Ruby also contains the binary executables of the hosting application software, which is used by the consumer for remote execution. In addition, the content providers have their representatives PA in the node Fire. Similar to the retailer, the content providers have their admin GUI located on Sky.

The database provider has its components ODBC Bridge Client spread on all the computing nodes that need data access (Fire, Ruby, Sky). The node Sun that actually hosts the database engine contains the ODBC Bridge Server and the native MS Access ODBC Driver. In addition, Sun runs the active server page enabled web server at the background. Therefore, Sun also needs the ODBC Bridge Client to be installed for the web server to query the database. As Sun does not need to contact

the other engineering objects, it does not contain the DPE.

The node Sky only hosts the administration interfaces for the three service providers (retailer, third party A and third party B). As the administration interfaces do not require the DPE, no ORB is needed on the node. Therefore, Sky is a non-DPE node.

There are two user domains, namely *Consumer I* and *Consumer II*. These two domains are running under two nodes Glacier and Mint. The two nodes contain two different operating systems for testing purposes. Consumer I occupies Glacier that runs Microsoft Windows NT and that contains ssUAP JAVA and MFC (c++) version. Consumer II possesses Mint, which runs Linux OS and has ssUAP JAVA and Qt (c++) version installed. Consumer I uses X-Win32 as the X-server while Consumer II uses XFree86 as the X-server. Both nodes contain a standard browser for the users to view their usage records through the WWW page.

6.4 Topology

This section discusses the physical and logical connections between the nodes. The physical connections between the six computers and two Ethernet switches are shown in figure 6.1.

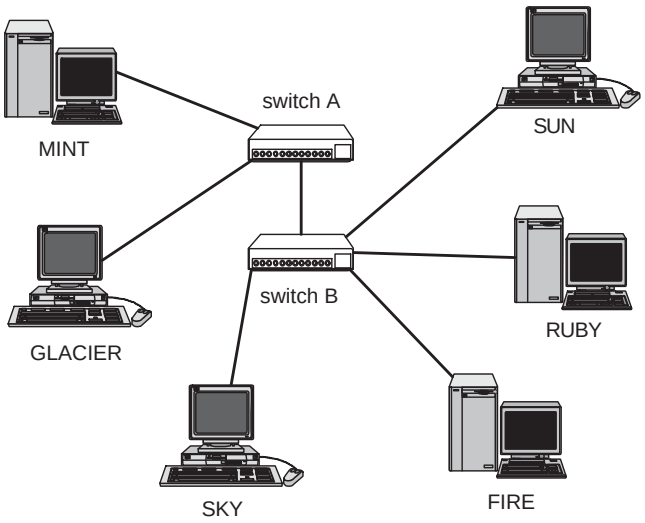


Figure 6.1: Topology

The connections are arranged to reflect the roles of the nodes. The consumer nodes (Mint and Glacier) are concentrated into switch A. The nodes that host the components of the service providers are connected to switch B. Although this may not

be the optimised configuration, it is simple to manage. If specific QoS requirements are imposed, the engineering objects can be relocated or the physical connections between the nodes can be changed to suit the requirements.

TINA separates the network connections into different layers. One of the purposes of these layers is to separate the service logic from the physical connection. Three network planes, namely *the Kernel Transport Network*, *the Transport Network* and *the Path Layer Network*, of the system configuration are shown in figure 6.2.

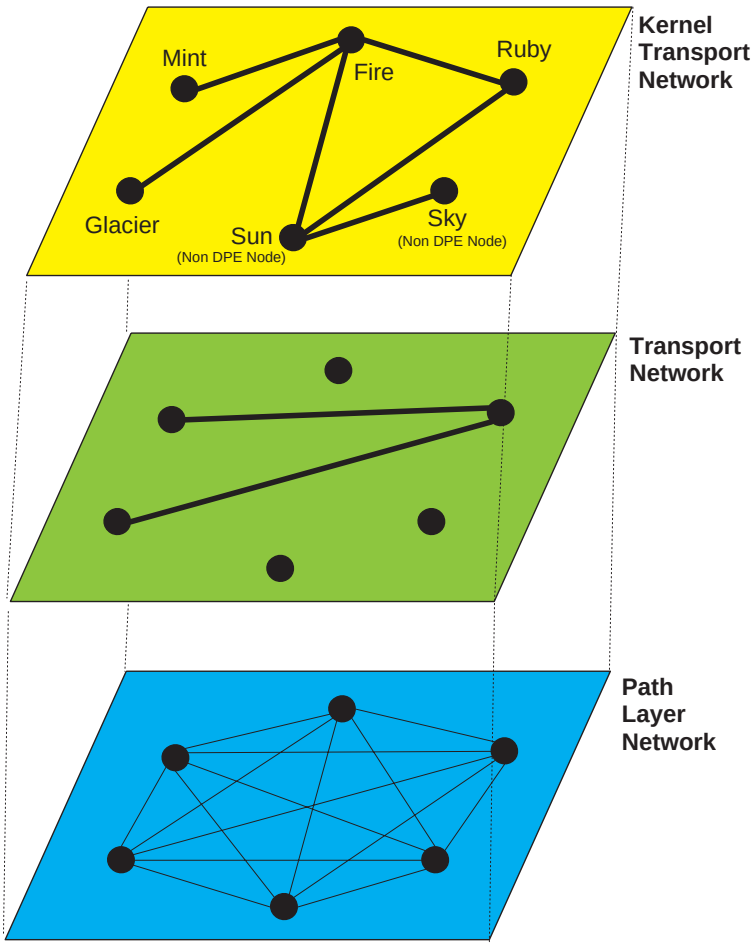


Figure 6.2: Logical Network Planes

Path Layer Network

The lowest plane is the Path Layer Network, which is related to the physical path of the network. The fully connected nodes on the plane show that the six computing nodes in the system can make a point-to-point direct connection path to one another. Physically the connections are made possible by going through the two Ethernet switches.

Transport Network

The middle plane is the Transport Network, which shows the stream binding connections. The middle plane in figure 6.2 shows that the two consumer nodes (Glacier and Mint) have direct stream binding connections to the third party content provider node (Ruby).

Kernel Transport Network

The top plane is the Kernel Transport Network, which shows the logical connections between the objects that exchange messages through the operational interfaces. One can notice that the consumer nodes (Glacier and Mint) only connect to the retailer node (Fire). This is logical because the consumers only obtain the service from the retailer. The engineering objects in the consumer nodes only invoke the operations on the engineering objects inside the retailer node. In this project, the Transport Network and the Kernel Transport Network use the same physical connections provided by the Path Layer Network.

The retailer node (Fire) has connections with the database provider node (Sun) and the third party content provider node (Ruby). These connections illustrate that they have business relationships.

The administration node (Sky) only has one connection to the database provider node (Sun). The communications between these two nodes do not pass through the DPE, as they are non-DPE nodes.

6.5 Applicable Application Software

Most Unix software packages[5] designed for use in the X environment can be used as the hosted application, providing that it is executable under the same machine that hosts the RSMSCore object. The software packages that cannot be used as the hosted application are the packages that use special X-server extensions, such as the "MIT-SHM"[6] Besides Unix packages, Java applications can also be used as the hosted applications. Unfortunately, Microsoft Windows executables are not designed

[5]Refer to Appendix C for a list of tested applications

[6]The "MIT-SHM" extension enables video memory to be *shared* between the application and the VGA card for fastest possible video data transfer. This is normally for video playback purpose. Normal application does not require this extension.

for use in the X environment. There are two solutions for this problem:

1. Use Windows Emulator under Unix environment.
2. Use Windows Terminal Server and have ssUAP integrated with the Citrix's Terminal Server Unix client.

The first solution has already been tested as workable in this project. However, the software packages that can be run under Windows Emulator are limited. Coupled with the performance and reliability factors, this solution is not very practical. The second solution needs the RSMSCore object to be ported into the Windows Terminal Server Environment. As there is insufficient available documentation about the Windows Terminal Environment, to be able to tightly integrate with the Terminal Server is very difficult, even if the source code of the Terminal Server is available. Nevertheless, the trend of the software industry is towards open source and platform independent, which may result in the requirement of running Microsoft Windows executable less and less important.

Chapter 7

Conclusion

7.1 Discussion

The RSMS is a realisation of some concepts of the TINA specifications. In building the RSMS service, some abstract TINA concepts become clearer and more understandable. It demonstrates the viability and flexibility of the TINA system. Besides the objective of building the RSMS service, the contributions of this project to the SATINA platform are the followings:

1. The third party related interfaces are defined for the platform. These interfaces complement the existing retailer interfaces to create something close to the third party reference points, which has not yet been undefined by TINA. The RSMS shows how to use these interfaces to activate, maintain and destroy third party sessions. These interfaces are service generic and have been used by other services on the SATINA platform.

2. The incorporation of ODBC Bridge components shows the use of non-DPE services in the TINA environment.

3. The RSMS shows the feasibility of the platform independence of the DPE by providing components that are designed for different platform. For example, the RSMS has JAVA, native Windows and native Linux client GUIs that use different software libraries. It demonstrates the way to use the service generic JAVA PA to initiate the native Windows ssUAP or native Linux ssUAP.

4. The usage records stored in the database can be used for testing the TINA billing service that may be created in future.

Apart from the contributions of the RSMS, the system has some the following limitations:

1. A few applications that uses special X-server extensions cannot be executed under remote X-servers, for example the MIT-SHM extension. In addition, Microsoft Windows applications have to be run under the Windows emulator, which results in performance degradation and unreliability.

2. The database provider is not DPE compliant, which make it difficult for the other services to use the database.

3. The startup sequence of the system is quite complicated. However, this cannot be avoided as the environment is much more complex than that of a single service provider.

7.2 Conclusion

An experimental ASP service, namely the *Remote Software Management Service* is designed and implemented. The TINA service architecture and computation architecture are used in designing and implementing of such service. The software developed in this project is deployed on the SATINA platform. The service offered by the RSMS is a collaboration of several service providers, which fellows the business model defined by TINA. The TINA concepts enable the RSMS system model to have advantages over most of the traditional ASP services. Some of the shortcomings of the existing ASP architecture are being addressed.

Quality of service - The QoS requirement is separated from the service design. Theoretically, the RSMS service provider can achieve the QoS requirements by requesting the connectivity provider to provide the adequate bandwidth. The standard connectivity reference points allow the RSMS service provider to easily switch to or aggregate with other compliant connectivity provider. This can be achieved once the required *network resource architecture* has been fully defined on the SATINA platform.

Interoperability between ASPs - The standard reference points enable the communication between the components of the content providers through the DPE. Therefore, the content providers can form alliance and operate together to achieve a common goal.

Compatibility between ASPs - The content providers have standard authentication mechanism, standard access sessions, standard database service and standard billing records. Adhering to the standard interfaces ensures the service providers to be compatible with each other.

Integration with third parties - The clear scope of responsibilities and predefined policies allow the cooperation of different parties with minimum confusion. The agreements on the reference points among different parties create standard interfaces. Such interfaces enable the retailer to make use of the third party services as if they were its own functions.

Weakness in certain fields - As the service is divided into smaller service providers, they can focus on their fields of expertises. Moreover, the service providers are fully compatible, which allows the users to try to use different service providers. The users can then choose the best one to use, before committing to a long-term contract with the service provider.

Security - Although security is presently not concerned in the RSMS prototype, it is relatively easy to integrate confidence providers and security modules into the RSMS system.

7.3 Future Work

This project only builds the prototype of the RSMS and is not suitable for practical use in the business environments. The followings are the areas that need further attention.

1. Apply the TINA management architecture - The administrator interfaces are currently interacting with the database provider directly without authenticate or verification. It is recommended to use the TINA management architecture or other management architecture to create a proper management environment and components.

2. Upgrade Non-DPE node to DPE node - The database provider uses the ODBC Bridge components, which are not DPE compliant. Other TINA engineering objects, except those specifically designed for use with ODBC Bridge, will not be able to communicate directly with the database provider. Two solutions are suggested: the first solution is to upgrade the entire database provider to the object oriented database, which support CORBA functions. The other

solution is to wrap the ODBC components with TINA objects and provide access interfaces to the other engineering object to use the ODBC components through the DPE.

3. Incorporate other service modules - other service modules can be developed and integrated into the RSMS system as third party service providers. For example, a printing service provider, which offers printing service for different content providers, can be designed. The service can be implemented as a personal printing manager or a commercial printing service, which enables a user to use the same printing service provider even if different RSMS third party content providers are used. This approach can be extended to other areas such as remote storage provider (similar to the Internet hard disk), fax service provider and so on.

4. Integrate the X-server in the ssUAP object - The X-server is presently a standalone commercial package. As there are open source pure JAVA X-server recently available, the X-server can be integrated in the ssUAP object. This leads to a more secure and consistent ssUAP. The applications running remotely from the content provider can then be displayed on a standard browser. By converting the ssUAP from JAVA application to JAVA applet, the client can then use a standard browser as the platform for the entire ssUAP. This results in a completely platform independent ssUAP.

5. Security enhancement - Currently the content provider spawns the application software under the same Unix user, which introduces security concerns. Improvement can be achieved by mapping the individual TINA users into different Unix users when accessing the third party domain.

6. TINA compliant billing system - The usage records stored in the database should be pushed up to the service generic billing service defined by TINA. The user should be able to review the usage records through the generic billing service instead of the Web interface provided by the RSMS service.

References

[1] C. Abarca, P. Farley, J. Forslow, J. C. Garcia, P. F. Hansen T. Hamada, S. Hogg, H. Kamata, L. Kristiansen, C. A. Licciardi, H.Mulder, E. Utsunomiya, and M. Yates. *Service Architecture*. TINA Consortium, June 16, 1997. Version 5.0.

[2] C. Abarca, P. Farley, J. C. Garcia, T. Hamada, P. F. Hansen, P. Hellemans, C. A. Licciardi, K. Nakashiro, and M. Yates. *Service Component Specification*. TINA Consortium, Jan 19, 1998. Version 1.0b.

[3] Martin Chapman and Stefano Montesi. *Overall Concepts and Principles of TINA*. TINA Consortium, Feb 17, 1995. Version 1.0.

[4] P. Farley and R. Minetti. *Ret Reference Point Specifications*. TINA Consortium, Jan 27, 1998. Version 1.0.

[5] C. Gerlach. *The ASP Revolution: Why Hosted Application Will Transform Business*. Mainspring eStrategy Report, January 2000.

[6] P. Graubmann, W. Hwang, M. Kudela, K. MacKinnon, N. Mercouroff, and N. Watanabe. *Engineering Modelling Concepts (DPE Architecture)*. TINA Consortium, December 1994.

[7] Y. Inocue, M. Lapierre, and C. Mossotto. *The TINA Book*. Prentice Hall Europe, 1999.

[8] Rational Software Coporation. *UML Notation Guide*, Sep 1, 1997. Version 1.1.

Appendix A

Sequence Diagrams

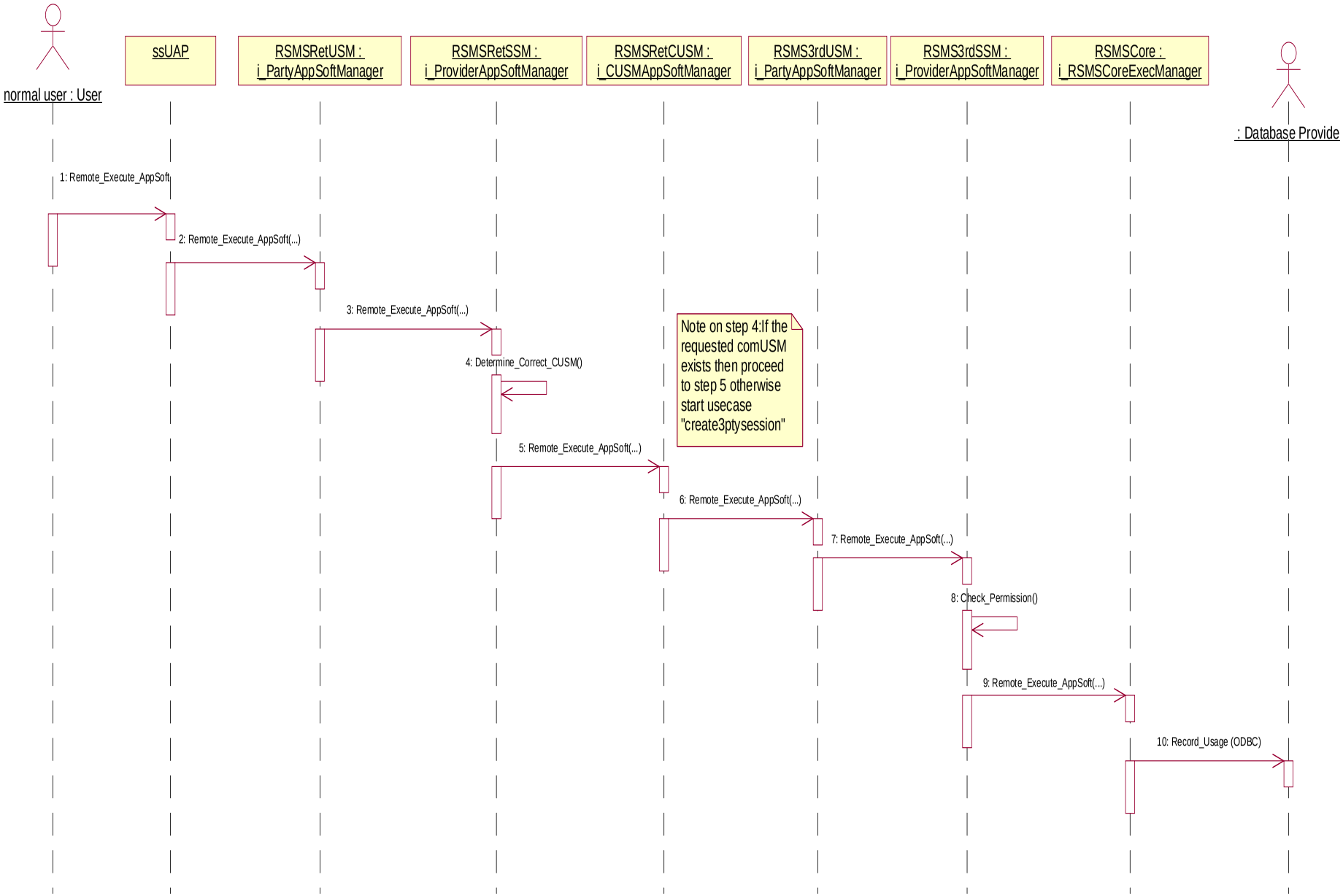


Figure A.1: Remote Execution

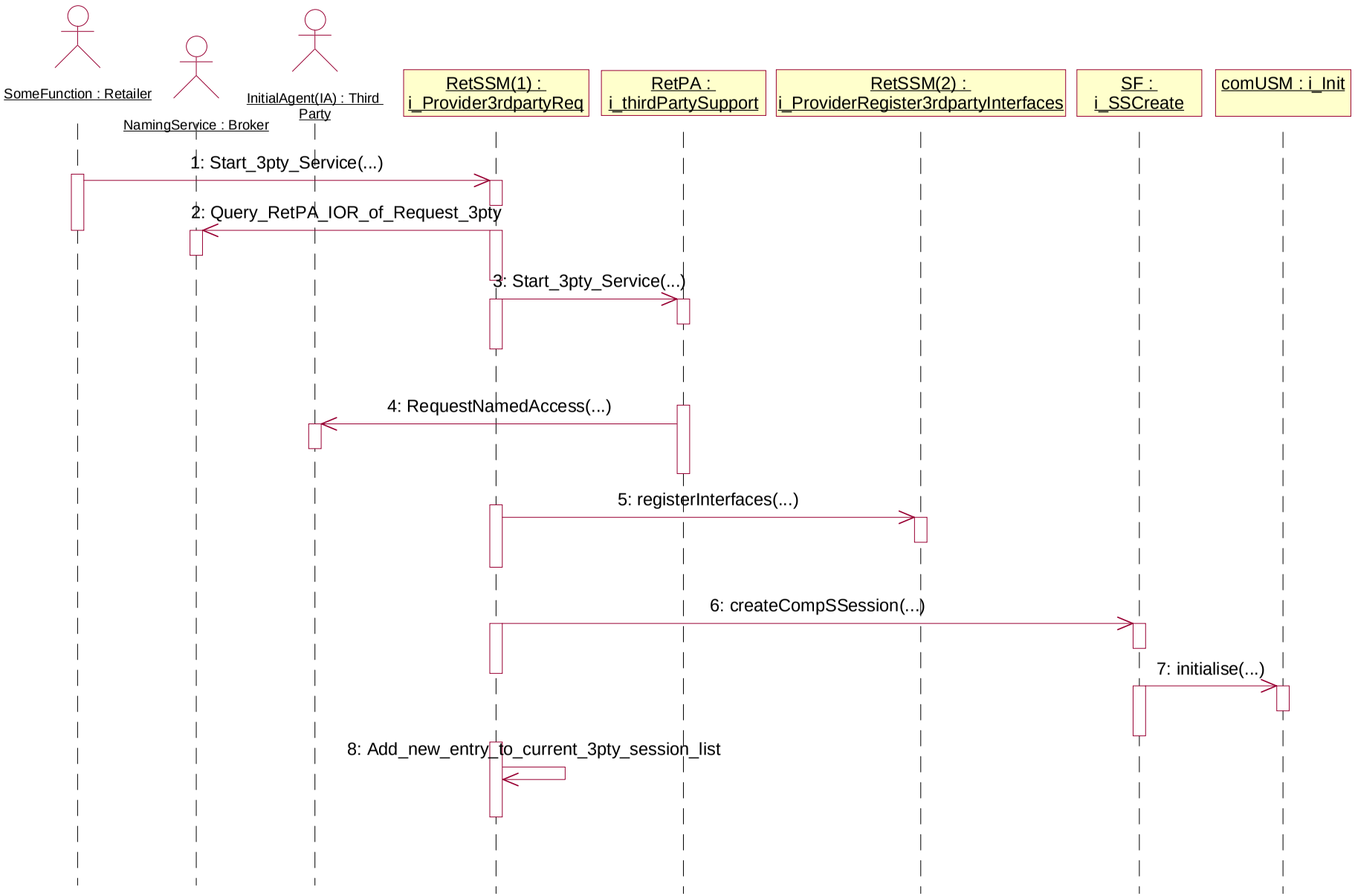


Figure A.2: Start Third Party Sessions

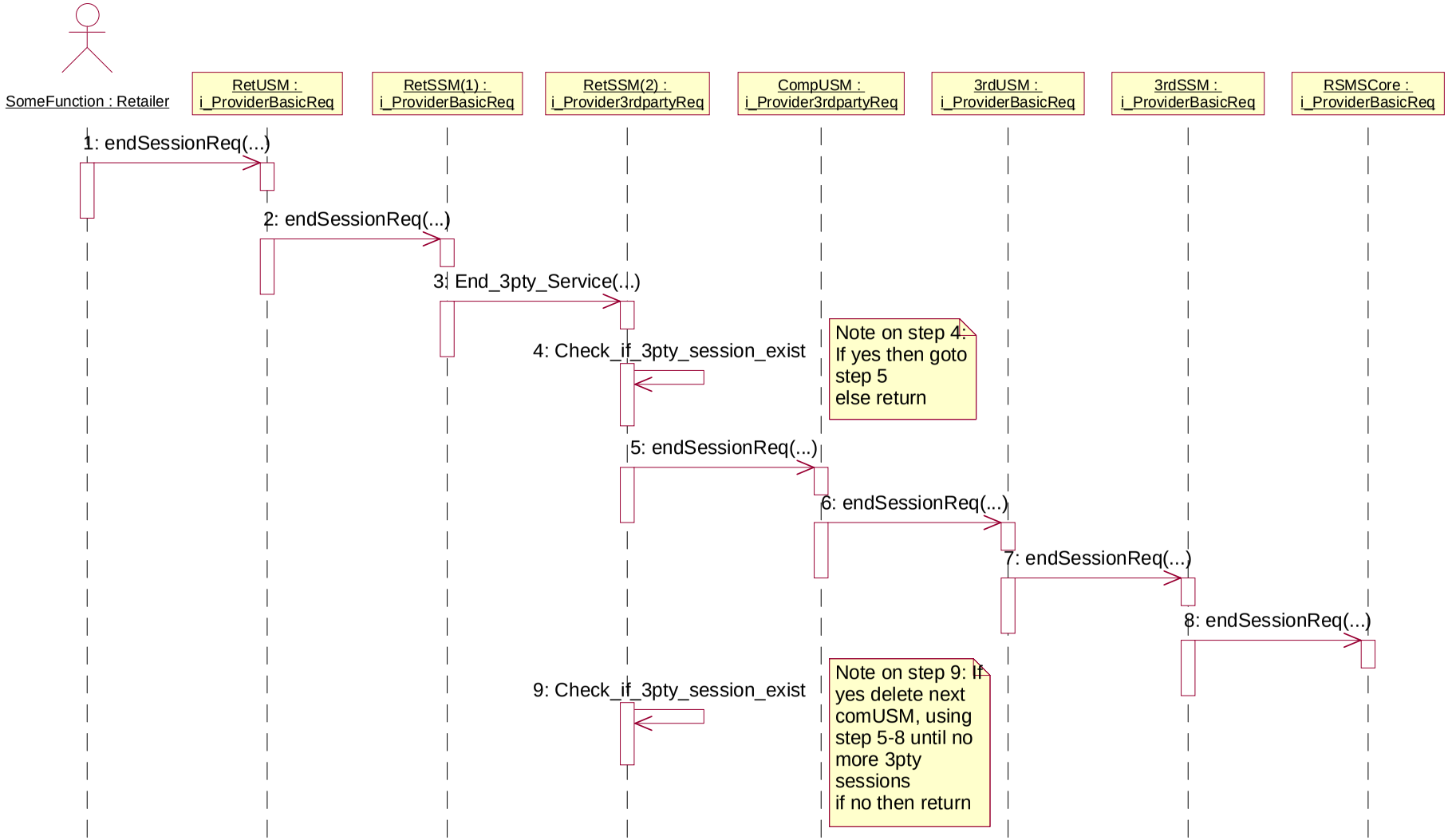


Figure A.3: Destroy Third Party Sessions

Appendix B

IDL definitions

B.1 RSMSCommonTypes

```
1  //Source file: Z:/RETAILER/IDL/RSMSTYPES/RSMSCOMMONTYPES.IDL
2
3  #ifndef __RSMSCOMMONTYPES_DEFINED
4  #define __RSMSCOMMONTYPES_DEFINED
5
6  /* CmIdentification
7    %X% %Q% %Z% %W% */
8
9  #include "TINACommonTypes.idl"
10
11 module RSMSCommonTypes {
12
13         typedef TINACommonTypes::t_PropertyList t_RSMSAppSoftProperties;
14
15         struct t_RSMSAppSoft {
16                 string name;
17                 t_RSMSAppSoftProperties properties;
18         };
19
20         typedef sequence <t_RSMSAppSoft> t_RSMSAppSoftList;
21
22 };
23
24 #endif
```

B.2 TINA3rdpartyCommonTypes

```
1
2  // File TINA3rdpartyCommonTypes.idl
3  // author: Chris Ip
4  // Description: For 3rd party usage Common. Not defined in TINA Specification
5  // Date: 16/08/2000
6  ////////////////////////////////////////////////////////////////////////
7
8  #ifndef tina3rdpartycommontypes_idl
9  #define tina3rdpartycommontypes_idl
10
11 #include "TINACommonTypes.idl"
12 #include "TINAAccessCommonTypes.idl"
13
14 module TINA3rdpartyCommonTypes {
15
16 enum t_3rdpartyUsageErrorCode {
17         UnknownUsageError,
18         ThirdPartyServerNotFound,    //
19         UsageNotAllowed,      // You don't have permission to do it
20         UsageNotAccepted,     // Owners have declined request
21         InvalidThirdPartyLogin,
22         ThridPartyServerBusy
23 };
24
25 exception e_3rdpartyUsageError {
26         t_3rdpartyUsageErrorCode errorCode;
27 };
28
29 typedef TINACommonTypes::Istring t_ThirdPtyName;
30
31 struct t_ThirdPtyInfo {
32         t_ThirdPtyName name;
33         TINACommonTypes::t_PropertyList properties;
34         TINACommonTypes::t_InterfaceList itfs;
35         TINAAccessCommonTypes::t_SessionInfo thirdpartySessionInfo;
36 };
37
```

```
38 typedef sequence<t_ThirdPtyInfo> t_ThirdPtyInfoList;
39
40 typedef TINACommonTypes::Istring t_ThirdPtyServerName;
41
42 }; //module TINA3rdpartyCommonTypes
43
44 #endif
```

B.3 TINA3rdptyInitial

```
1  //author: Chris Ip
2  //date:  23/06/2000
3  //description: For third party services
4  //this is not in TINA Specification and is SATINA specific
5
6  #ifndef __TINA3rdptyInitial_DEFINED
7  #define __TINA3rdptyInitial_DEFINED
8
9
10 #include "TINACommonTypes.idl"
11 #include "TINAAccessCommonTypes.idl"
12 #include "TINAProviderAccess.idl"
13 #include "TINA3rdpartyCommonTypes.idl"
14
15 module TINA3rdptyInitial {
16
17 interface i_thirdPartySupport {
18
19        void Start_3pty_Service (
20               out TINAAccessCommonTypes::t_SessionInfo sessionInfo,
21               in TINAAccessCommonTypes::t_ServiceId serviceId,
22               in TINACommonTypes::t_UserId UserID,
23               in TINACommonTypes::t_UserProperties UserProps,
24               in TINAProviderAccess::t_ApplicationInfo app
25        ) raises (
26               TINA3rdpartyCommonTypes::e_3rdpartyUsageError
27        );
28
29        void End_3pty_Service (
30               in TINAAccessCommonTypes::t_SessionInfo sessionInfo
31        ) raises (
32               TINA3rdpartyCommonTypes::e_3rdpartyUsageError
33        );
34
35 };
36
37
```

```
38 };
39 #endif
```

B.4 TINAProvider3rdpartyUsage

```
1
2  // File TINAProvider3rdpartyUsage.idl
3  // author: Chris Ip
4  // Description:For 3rd party usage. SATINA specific
5  ////////////////////////////////////////////////////////
6  // author: Chris Ip
7  // Date:  10/08/2000
8  // Description: Add Multiple 3rd party functions
9  //
10 ////////////////////////////////////////////////////////
11 // author: Chris Ip
12 // Date:  16/08/2000
13 // Description: Add Exceptions for error handling
14 ////////////////////////////////////////////////////////
15
16 #ifndef tinaprovider3rdpartyusage_idl
17 #define tinaprovider3rdpartyusage_idl
18
19 #include "TINACommonTypes.idl"
20 #include "TINAUsageCommonTypes.idl"
21 #include "TINAAccessCommonTypes.idl"
22 #include "TINAProviderAccess.idl"
23 #include "TINA3rdpartyCommonTypes.idl"
24
25 module TINAProvider3rdpartyUsage {
26
27 interface i_ProviderRegister3rdpartyInterfaces {
28
29        void registerInterfaces (
30               in TINACommonTypes::t_ParticipantSecretId myId,
31               inout TINACommonTypes::t_RegisterInterfaceList itfs,
32               in TINA3rdpartyCommonTypes::t_ThirdPtyServerName ServerName
33        ) raises (
34               TINAUsageCommonTypes::e_UsageError,
35               TINACommonTypes::e_InterfacesError,
36               TINACommonTypes::e_RegisterError
37        );
```

```
};      // interface i_Provider3rdpartyInterfaces

interface i_Provider3rdpartyReq {

        void Start_3pty_Service (
                in TINACommonTypes::t_SessionId SessionId,
                in TINACommonTypes::t_UserId UserID,
                in TINAProviderAccess::t_ApplicationInfo app,
                in TINAAccessCommonTypes::t_ServiceId serviceId,
                in TINACommonTypes::t_UserId thirdptyLogin,
                in TINACommonTypes::t_UserProperties thridptyProps
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
        );

        void End_3pty_Service (
                in TINACommonTypes::t_SessionId SessionId,
                in TINACommonTypes::t_ParticipantSecretId SecretId,
                in TINA3rdpartyCommonTypes::t_ThirdPtyServerName ServerName
        ) raises (
                TINAUsageCommonTypes::e_UsageError,
                TINA3rdpartyCommonTypes::e_3rdpartyUsageError
        );

};      // interface i_Provider3rdpartyReq



};      // module TINAProvider3rdpartyUsage


#endif
```

B.5 RSMSRetUSM

```
1  //Source file: Z:/RETAILER/IDL/RSMSRET/RSMSRETUSM.IDL
2
3  #ifndef __RSMSRETUSM_DEFINED
4  #define __RSMSRETUSM_DEFINED
5
6  /* CmIdentification
7    %X% %Q% %Z% %W% */
8
9  #include "../RSMSTypes/RSMSCommonTypes.idl"
10 #include "TINACommonTypes.idl"
11
12 module RSMSRetUSM {
13
14 interface i_PartyAppSoftManager {
15
16         void Request_Software_List (
17                 out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
18                 in TINACommonTypes::t_SessionId SessionId,
19                 in TINACommonTypes::t_UserId UserID,
20                 in TINACommonTypes::t_UserProperties UserProps,
21                 in boolean UpdateList
22         );
23
24         void Batch_Download (
25                 in RSMSCommonTypes::t_RSMSAppSoftList aList,
26                 in TINACommonTypes::t_SessionId SessionId,
27                 in TINACommonTypes::t_UserId UserID,
28                 in TINACommonTypes::t_UserProperties UserProps
29         );
30
31         void Remote_Execute_AppSoft (
32                 in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
33                 in TINACommonTypes::t_SessionId SessionId,
34                 in TINACommonTypes::t_UserId UserID,
35                 in TINACommonTypes::t_UserProperties UserProps
36         );
37
```

```
38        void Local_Execute_AppSoft (
39                in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
40                in TINACommonTypes::t_SessionId SessionId,
41                in TINACommonTypes::t_UserId UserID,
42                in TINACommonTypes::t_UserProperties UserProps
43        );
44
45 };
46
47 interface i_PartyAdminAppSoftManager {
48
49        void Add_AppSoft (
50                in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
51                in TINACommonTypes::t_UserId UserID,
52                in TINACommonTypes::t_UserProperties UserProps
53        );
54
55        void Del_AppSoft (
56        in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
57                in TINACommonTypes::t_UserId UserID,
58                in TINACommonTypes::t_UserProperties UserProps
59        );
60
61        void  List_AppSoft(
62                out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
63                in TINACommonTypes::t_UserId UserID,
64                in TINACommonTypes::t_UserProperties UserProps
65    );
66
67 };
68
69 };
70 #endif
```

B.6 RSMSRetSSM

```
1
2  #ifndef __RSMSRETSSM_DEFINED
3  #define __RSMSRETSSM_DEFINED
4
5
6  #include "../RSMSTypes/RSMSCommonTypes.idl"
7  #include "TINACommonTypes.idl"
8
9  module RSMSRetSSM {
10
11 interface i_ProviderAppSoftManager {
12
13         void Request_Software_List (
14                 out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
15                 in TINACommonTypes::t_SessionId SessionId,
16                 in TINACommonTypes::t_UserId UserID,
17                 in TINACommonTypes::t_UserProperties UserProps,
18                 in boolean UpdateList
19         );
20
21         void Batch_Download (
22                 in RSMSCommonTypes::t_RSMSAppSoftList aList,
23                 in TINACommonTypes::t_SessionId SessionId,
24                 in TINACommonTypes::t_UserId UserID,
25                 in TINACommonTypes::t_UserProperties UserProps
26         );
27
28         void Remote_Execute_AppSoft (
29                 in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
30                 in TINACommonTypes::t_SessionId SessionId,
31                 in TINACommonTypes::t_UserId UserID,
32                 in TINACommonTypes::t_UserProperties UserProps
33         );
34
35         void Local_Execute_AppSoft (
36                 in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
37                 in TINACommonTypes::t_SessionId SessionId,
```

```
38              in TINACommonTypes::t_UserId UserID,
39              in TINACommonTypes::t_UserProperties UserProps
40      );
41
42 };
43
44 interface i_ProviderAdminAppSoftManager {
45
46      //the administrators interface for adding and removing appsoft
47      //from the server should not be exported accross the retailer
48      //interface this interface is to be exported to the USM
49
50
51      void Add_AppSoft (
52              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
53              in TINACommonTypes::t_UserId UserID,
54              in TINACommonTypes::t_UserProperties UserProps
55      );
56
57      void Del_AppSoft (
58              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
59              in TINACommonTypes::t_UserId UserID,
60              in TINACommonTypes::t_UserProperties UserProps
61      );
62
63      void List_AppSoft(
64              out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
65              in TINACommonTypes::t_UserId UserID,
66              in TINACommonTypes::t_UserProperties UserProps
67
68      );
69 };
70 };
71 #endif
```

B.7 RSMSCUSM

```
1  //Source file: Z:/RETAILER/IDL/RSMSRET/RSMSCUSM.IDL
2
3  #ifndef __RSMSCUSM_DEFINED
4  #define __RSMSCUSM_DEFINED
5
6  /* CmIdentification
7    %X% %Q% %Z% %W% */
8
9  #include "../RSMSTypes/RSMSCommonTypes.idl"
10 #include "TINACommonTypes.idl"
11
12 module RSMSCUSM {
13
14 interface i_CUSMAppSoftManager {
15
16         void Request_Software_List (
17                 out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
18                 in TINACommonTypes::t_SessionId SessionId,
19                 in TINACommonTypes::t_UserId UserID,
20                 in TINACommonTypes::t_UserProperties UserProps,
21                 in boolean UpdateList
22         );
23
24         void Batch_Download (
25                 in RSMSCommonTypes::t_RSMSAppSoftList aList,
26                 in TINACommonTypes::t_SessionId SessionId,
27                 in TINACommonTypes::t_UserId UserID,
28                 in TINACommonTypes::t_UserProperties UserProps
29         );
30
31         void Remote_Execute_AppSoft (
32                 in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
33                 in TINACommonTypes::t_SessionId SessionId,
34                 in TINACommonTypes::t_UserId UserID,
35                 in TINACommonTypes::t_UserProperties UserProps
36
37         );
```

```
38
39          void Local_Execute_AppSoft (
40                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
41                  in TINACommonTypes::t_SessionId SessionId,
42                  in TINACommonTypes::t_UserId UserID,
43                  in TINACommonTypes::t_UserProperties UserProps
44          );
45
46  };
47
48  interface i_CUSMAdminAppSoftManager {
49
50          //the administrators interface for adding and removing appsoft
51          //from the server should not be exported accross the retailer
52          //interface this interface is to be exported to the USM
53          void Add_AppSoft (
54                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
55                  in TINACommonTypes::t_UserId UserID,
56                  in TINACommonTypes::t_UserProperties UserProps
57          );
58
59          void Del_AppSoft (
60                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
61                  in TINACommonTypes::t_UserId UserID,
62                  in TINACommonTypes::t_UserProperties UserProps
63          );
64
65          void  List_AppSoft(
66                  out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
67                  in TINACommonTypes::t_UserId UserID,
68                  in TINACommonTypes::t_UserProperties UserProps
69
70          );
71  };
72  };
73  #endif
```

B.8 RSMS3rdUSM

```
1  //Source file: Z:/RETAILER/IDL/RSMSRET/RSMS3rdUSM.IDL
2
3  #ifndef __RSMS3RDUSM_DEFINED
4  #define __RSMS3RDUSM_DEFINED
5
6  #include "../RSMSTypes/RSMSCommonTypes.idl"
7  #include "TINACommonTypes.idl"
8
9  module RSMS3rdUSM {
10
11 interface i_PartyAppSoftManager {
12
13 //behaviour: this interface is to be exported to the client PA
14
15        void Request_Software_List (
16               out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
17               in TINACommonTypes::t_SessionId SessionId,
18               in TINACommonTypes::t_UserId UserID,
19               in TINACommonTypes::t_UserProperties UserProps,
20               in boolean UpdateList
21               );
22
23        void Batch_Download (
24               in RSMSCommonTypes::t_RSMSAppSoftList aList,
25               in TINACommonTypes::t_SessionId SessionId,
26               in TINACommonTypes::t_UserId UserID,
27               in TINACommonTypes::t_UserProperties UserProps
28               );
29
30        void Remote_Execute_AppSoft (
31               in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
32               in TINACommonTypes::t_SessionId SessionId,
33               in TINACommonTypes::t_UserId UserID,
34               in TINACommonTypes::t_UserProperties UserProps
35
36        );
37
```

```
38          void Local_Execute_AppSoft (
39                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
40                  in TINACommonTypes::t_SessionId SessionId,
41                  in TINACommonTypes::t_UserId UserID,
42                  in TINACommonTypes::t_UserProperties UserProps
43          );
44
45  };
46
47  interface i_PartyAdminAppSoftManager {
48
49          //the administrators interface for adding and removing appsoft
50          //from the server should not be exported accross the retailer
51          //interface this interface is to be exported to the USM
52
53          void Add_AppSoft (
54                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
55                  in TINACommonTypes::t_UserId UserID,
56                  in TINACommonTypes::t_UserProperties UserProps
57          );
58
59          void Del_AppSoft (
60                  in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
61                  in TINACommonTypes::t_UserId UserID,
62                  in TINACommonTypes::t_UserProperties UserProps
63          );
64
65          void  List_AppSoft(
66                  out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
67                  in TINACommonTypes::t_UserId UserID,
68                  in TINACommonTypes::t_UserProperties UserProps
69
70          );
71  };
72  };
73  #endif
```

B.9 RSMS3rdSSM

```
1
2  #ifndef __RSMS3RDSSM_DEFINED
3  #define __RSMS3RDSSM_DEFINED
4
5
6  #include "../RSMSTypes/RSMSCommonTypes.idl"
7  #include "TINACommonTypes.idl"
8
9  module RSMS3rdSSM {
10
11 interface i_ProviderAppSoftManager {
12
13 //behaviour: this interface is to be exported to USM within the retailer domain
14
15        void Request_Software_List (
16               out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
17               in TINACommonTypes::t_SessionId SessionId,
18               in TINACommonTypes::t_UserId UserID,
19               in TINACommonTypes::t_UserProperties UserProps,
20               in boolean UpdateList
21               );
22
23        void Batch_Download (
24               in RSMSCommonTypes::t_RSMSAppSoftList aList,
25               in TINACommonTypes::t_SessionId SessionId,
26               in TINACommonTypes::t_UserId UserID,
27               in TINACommonTypes::t_UserProperties UserProps
28               );
29
30        void Remote_Execute_AppSoft (
31               in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
32               in TINACommonTypes::t_SessionId SessionId,
33               in TINACommonTypes::t_UserId UserID,
34               in TINACommonTypes::t_UserProperties UserProps
35        );
36
37        void Local_Execute_AppSoft (
```

```
38              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
39              in TINACommonTypes::t_SessionId SessionId,
40              in TINACommonTypes::t_UserId UserID,
41              in TINACommonTypes::t_UserProperties UserProps
42      );
43
44 };
45
46 interface i_ProviderAdminAppSoftManager {
47
48      //the administrators interface for adding and removing appsoft
49      //from the server should not be exported accross the retailer
50      //interface this interface is to be exported to the USM
51
52      void Add_AppSoft (
53              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
54              in TINACommonTypes::t_UserId UserID,
55              in TINACommonTypes::t_UserProperties UserProps
56      );
57
58      void Del_AppSoft (
59              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
60              in TINACommonTypes::t_UserId UserID,
61              in TINACommonTypes::t_UserProperties UserProps
62      );
63
64      void List_AppSoft(
65              out RSMSCommonTypes::t_RSMSAppSoftList RequiredList,
66              in TINACommonTypes::t_UserId UserID,
67              in TINACommonTypes::t_UserProperties UserProps
68    );
69 };
70 };
71 #endif
```

B.10 RSMSCore

```
1  //author: Chris Ip
2  //date: 20-05-2000
3  //description: idl for the CORE functions of RSMS service
4
5  #ifndef __RSMSCORE_DEFINED
6  #define __RSMSCORE_DEFINED
7
8  #include "../RSMSTypes/RSMSCommonTypes.idl"
9  #include "TINACommonTypes.idl"
10
11 module RSMSCore {
12
13 //3rd party provider could choose to support either interfaces or both
14 //this is why request software list appears on both interfaces
15
16
17 interface i_RSMSCoreDownloadManager {
18
19         void Request_Software_List (
20                 out RSMSCommonTypes::t_RSMSAppSoftList RequestedList,
21                 in TINACommonTypes::t_SessionId SessionId,
22                 in TINACommonTypes::t_UserId UserID,
23                 in TINACommonTypes::t_UserProperties UserProps,
24                 in boolean UpdateList
25         );
26
27         void Batch_Download (
28                 in RSMSCommonTypes::t_RSMSAppSoftList aList,
29                 in TINACommonTypes::t_SessionId SessionId,
30                 in TINACommonTypes::t_UserId UserID,
31                 in TINACommonTypes::t_UserProperties UserProps
32         );
33 };
34
35 interface i_RSMSCoreExecManager {
36
37         void Request_Software_List (
```

```
38              out RSMSCommonTypes::t_RSMSAppSoftList RequestedList,
39              in TINACommonTypes::t_SessionId SessionId,
40              in TINACommonTypes::t_UserId UserID,
41              in TINACommonTypes::t_UserProperties UserProps,
42              in boolean UpdateList
43      );
44
45      void Remote_Execute_AppSoft (
46              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
47              in TINACommonTypes::t_SessionId SessionId,
48              in TINACommonTypes::t_UserId UserID,
49              in TINACommonTypes::t_UserProperties UserProps
50      );
51 };
52
53 interface i_RSMSCoreAdmin{
54
55      //the administrators interface for adding and removing appsoft
56      //from the server should not be exported accross the retailer
57      //interface this interface is to be exported to the USM
58      void Add_AppSoft (
59              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
60              in TINACommonTypes::t_UserId UserID,
61              in TINACommonTypes::t_UserProperties UserProps
62      );
63
64      void Del_AppSoft (
65              in RSMSCommonTypes::t_RSMSAppSoft _AppSoft,
66              in TINACommonTypes::t_UserId UserID,
67              in TINACommonTypes::t_UserProperties UserProps
68      );
69
70      void List_AppSoft(
71              out RSMSCommonTypes::t_RSMSAppSoftList RequredList,
72              in TINACommonTypes::t_UserId UserID,
73              in TINACommonTypes::t_UserProperties UserProps
74
75      );
76 };
```

```
77 };
78 #endif
```

Appendix C

Tested Applications

Acrobat Reader (Windows, ver 4)
Freecell (Windows, ver 4.10)
Gedit (Linux, ver 0.5.4)
gFTP (Linux,ver 2.04)
GIMP (Linux, ver 1.04)
Gnome Calender (Linux, ver 1.0.10)
GnomeCard (Linux)
Gnumeric (Linux, ver 0.35)
Gqview(Linux, ver 0.7.0)
Gtop (Linux, ver 1.0.3)
Hearts (Windows, ver 4.10)
Karchie (Linux, ver 1.1.2)
Kcalc (Linux)
Kdvi (Linux, ver 0.4.3)
Kedit (Linux, ver 1.2.2)
Kfind (Linux, ver 0.4.1)
Kfinger (Linux, ver 0.8.2)
Kghostview (Linux, ver 0.7)
Kmahjongg (Linux, ver 0.4.1)
Kmail (Linux, ver 1.0.28)
Kmines (Linux, ver 1.0.1a)
Korginizer (Linux, ver 1.1.1)
Kpackage (Linux, ver 1.3.8)
Kpaint (Linux, ver 0.4.3)
Kpoker(Linux, ver 0.5)
Kreversi (Linux, ver 1.0.1)
Ksokoban (Linux, ver 0.2.2)
Ktop (Linux, ver 1.0.1)
Kview (Linux, ver 1.8)
KVt (Linux, ver 1.1.1.1)
Kwrite (Linux, ver 0.98)
Lotus Organizer (Windows)
Minesweeper (Windows, ver 4.10)
MpegTV(Linux, ver 1.1.1.1) video only
MSPaint (Windows, ver 4.10)
NotePad(Windows, ver 4.10)
Solitaire (Windows, ver 4.10)
Star Office(Linux, ver 5.1)
WordPad (Windows, ver 4.10)
Xcalc (Linux)
Xclock (Linux)
Xdvi (Linux, ver 22.05d-k)
Xpaint (Linux, ver 2.4.9)
Xview (Linux, ver 4.1)

[Place Bibliography Here]