EVALUATION OF WIDGET-BASED APPROACHES FOR DEVELOPING RICH INTERNET APPLICATIONS

Paco Azevedo Mendes

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

Johannesburg, 2010

Declaration

I declare that this research report is my own, unaided work, other than where specifically 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 30 day of August 2010

Paco Azevedo Mendes

Abstract

A widget is a packaged interactive client-side application developed using Web standards and techniques. Widgets present an opportunity to formalise the approaches that are used to develop and deploy Rich Internet Applications (RIAs) for multiple platforms. The objective of the research presented in this report is to evaluate whether widgets successfully meet the requirements of a RIA, which are identified by behaviour, ease of development, security, portability and presentation concerns. A review of the widget landscape has been conducted and used to develop a conceptual widget framework. A film rating and recommendation service has been developed based on this framework and is used to evaluate whether widgets meet the requirements of a RIA. It has been determined that widget-based development approaches assist in reducing the complexities of RIA development. This is achieved by providing developers with simplified interfaces that abstract the complexities of accessing local and remote behaviour. The main weaknesses identified are the vast incompatibilities between existing implementations and the relaxed security measures being used. Further standardisation efforts are required to reduce the incompatibilities in the landscape and improve the security and portability of widgets.

Para a Mãe, Pai, Marta, Luca, Tomé, Leandra e Cocá

Acknowledgements

I would like to thank and acknowledge the support received from my supervisor Professor Dwolatzky, the sponsors of the Centre for Telecommunications Access and Services at the University of the Witwatersrand and my personal sponsor, Standard Bank. To Marcos Cáceres at Opera Software and the W3C, thank you for sharing your knowledge and insight into widgets and the W3C standardisation activities.

Contents

List of Figures

List of Tables

List of Abbreviations

AIR: Adobe Integrated Runtime

API: Application Programming Interface

CHTML: Compressed Hypertext Markup Language

COM: Component Object Model

CSS: Cascading Style Sheet

DOM: Document Object Model

XML: eXtensible Markup Language

GIF: Graphic Interchange Format

GWT: Google Web Toolkit

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IMDB: Internet Movie Database

IP: Internet Protocol

IT: Information Technology

ITV: Interactive Television

JIL: Joint Innovation Lab

JPEG: Joint Photographic Experts Group

OMTP: Open Mobile Terminal Platform

PNG: Portable network Graphics

RBAC: Role-Based Access Control

RIA: Rich Internet Application

SDK: Software Development Kit

SDP: Service Delivery Platform

SOA: Service Oriented Architecture

UI: User Interface

URI: Universal Resource Identifier

VM: Virtual Machined

W3C: World Wide Web Consortium

WAP: Wireless Application Protocol

WARP: Widget Access Request Policy

WWW: World Wide Web

Chapter 1

Introduction

The extensive adoption and use of the Internet and World Wide Web (WWW)-based services continues to significantly influence the broader software development landscape. The Web serves both as a popular and strategic environment to deploy new software systems capable of delivering applications on different platforms [1–3]. Many developers are using the Web as an applications environment, and many views exist for what a *Web application* is. The Internet and Web have undergone a number of evolutionary phases to support a growing desire for richer networked applications that allow for: e-commerce, multimedia delivery, social networking, navigation, gaming, desktop style collaborative word processor and spreadsheet type applications. This sustained demand for multi-platform rich Internet applications (RIA), continues to encourage the use of new technologies to meet an ever-increasing scope of requirements [4].

Although the Web browser continues to serve as the primary means of interacting with the Web, a growing need to combine services provided on consumer platforms with a range of Web-based services, has given rise to a new class of application known as a *widget*. Unlike a Web page, which is accessed using a Web browser, a widget is a full-fledged interactive application hosted on a client and developed using Web standards. Widgets typically have simple interfaces, yet they can be used to provide rich personalised applications by integrating the capabilities of a host device with those of Web applications.

1.1 A Brief History of Widgets

Since the inception of the Web on the Internet, a number of new methods and techniques have surfaced to allow users to interact with distributed content and applications. As the Web has evolved and grown, so too have the number of platforms and interfaces allowing users to interact with resources on the Web.

1.1.1 The Evolution of the Web

As described by Tim Berners-Lee [5], the Web has rapidly grown from a simple platform allowing users to share and link text-based documents, to an advanced platform supporting rich, intelligent Web applications. The Web of documents has become a Web of applications and is moving towards a semantic *Web of things* [5, 6].

In its infancy, the Web provided a useful means of being able to link any document to any other through the use of hyperlinks and the hypertext transfer protocol (HTTP). As the base of Web users grew, so too did a desire for new features and capabilities. Web servers and browsers rapidly advanced to support many new requirements to enable visually attractive and interactive interfaces for the wide range of applications made accessible by the Internet. New technologies and protocols surfaced and improvements were made to the hypertext markup language (HTML), universal resource identifiers (URIs), Web graphics, Cascading Style Sheets (CSS), the Document Object Model (DOM) and client-side scripting languages such as JavaScript. E-commerce promoted the use of form fields and the advancement of encryption and reliable security models [7]. A number of browser plug-ins, such as Flash, have been introduced to support richer multimedia interfaces and video [8]. Web interfaces have improved to provide a broader base of users with the ability to publish content through the use of wiki's, blogs and other content management systems, that abstract the underlying markup and complexities of Web documents [9]. The eXtensible Markup Language (XML) has become an essential standard that enables and ensures a uniform exchange and definition of data used by different Information Technology (IT) systems interacting with the Web [5]. Different forms of Web services have been created to allow for greater machine to machine interactions and the exchange of data between different software systems and organisations [10]. All these advances of the Web support a common goal, to allow the Web to scale and let any user on any platform access content and applications in a consistent way [5].

A more recent development on the Web is the use of the JavaScript `XMLHttpRequest` object that allows scripting code running in a Web browser to make asynchronous calls to a Web server. This allows a client to perform multiple background tasks and obtain selected data sets without refreshing an entire Web page [11]. This phase has seen the introduction of more responsive *Web 2.0* user interfaces that allow Web pages to behave in a more similar fashion to desktop applications. Web 2.0 does not signify a new version of the Web, but rather a new way in which existing Web technologies are used to build RIAs [4]. Asynchronous JavaScript and XML (Ajax) is a popular term used to identify the group of technologies that allow for these responsive Web-based applications to run within Web browsers [11]. Ajax has assisted in establishing JavaScript as a dominant client-side scripting language integrated within Web browsers, however, Ajax implementations are not restricted to JavaScript and XML. Ajax has introduced a new philosophy for developing applications that further blur the boundary between desktop and browser interfaces. Prominent applications developed by Google, Microsoft, Yahoo! and Facebook, for instance, rely extensively on Ajax techniques, as do many modern Web sites.

A prominent proposal likely to influence the future use of the Web is the HTML5 draft recommendation being developed by the World Wide Web Consortium (W3C). The specification acknowledges the poor support that HTML provides for the demands of modern Web applications. The specification hopes to rectify this by updating the current HTML standard to support the current and future uses of the Web. HTML5 will gradually introduce new elements and Application Programming Interfaces (APIs) to improve interactivity and natively support features that include audio, video and data persistence [12].

1.1.2 Interfaces to Web Applications

Throughout the extensive and rapid evolution of the Web, the Web browser continues to remain the principal application, allowing users to access and interact with Web documents and applications. Browsers implement open Web standards and their usefulness results from users on different platforms having the ability to interact with different resources on the Web in a uniform way [3]. The core architecture of Web browsers, has however hardly changed since Tim Berners-Lee conceptualised them nearly twenty years ago [5]. The role of a browser is still primarily to allow users to navigate and browse various resources (documents and applications) on the Web from any platform. Web browsers are designed to run in sandbox environments on stationary devices, where Web content has very restricted access

to data and resources on a client. Even the Web browsers that have been migrated to mobile devices, continue to provide functionality comparable with their desktop counterparts [13].

As standards and protocols for the Internet and Web are well established and defined, it is possible that any application can be created to interact with the Web using the Internet. A number of alternative approaches exist for developing what is broadly described as a RIA:

> An Internet-based application that offers similar features and functionality to that of applications hosted on a client [4].

RIAs follow a conventional client-server architecture that comprises of clients to handle user behaviour and application servers to process and store data [1, 2, 11]. To reduce the complexity of developing RIAs, various frameworks are available. RIA frameworks are often based on Ajax principles and applications may or may not run within a Web browser [14]. Some of the prominent approaches to develop RIAs include:

- **Ajax-Based:** Microsoft ASP.Net Ajax and Google Web Toolkit [15, 16].
- **Proprietary-Based:** Adobe Flash, Microsoft Silverlight and Sun's (now Oracle's) JavaFX [8, 15, 17].
- **Mixed:** Adobe Integrated Runtime (AIR) which supports Ajax, and proprietary Flash or Flex [18].

Proprietary-based approaches typically require a browser plug-in or runtime environment installed on the client, whereas Ajax-based approaches normally only use client-side JavaScript natively supported by a Web browser.

1.1.3 The Rise of Widgets

Widgets have gained substantial attention as an alternative approach to developing and deploying RIAs on multiple service delivery platforms[1]. Widgets provide simple interfaces to services on the host device, such as clocks, battery gauges, notepads

[1] A service delivery platform is defined here as any network-connected device capable of delivering a service that consists of any combination of audio, video or data to a user.

(a) Weather forecast (b) Stock price feed (c) Battery gauge

Figure 1.1: Sample widget applications for the Yahoo! Konfabulator engine [20]

and application controllers; or interfaces to Web-based services such as news and weather feeds, currency exchange rates, email and Web photo albums. A user is able to regularly interact with a personalised inventory of widgets acquired from an extensive collection of both commercial and freely available widgets [19]. Screenshots of sample widgets are illustrated in figure 1.1.

Aside from the economic drivers of widget solutions, the inception of widgets is largely driven by a requirement to provide simple interfaces to regularly access personalised applications without the need for a full-fledged Web browser. Web browsers have become an immensely complicated application environment and widget approaches may be considered to be a lighter alternative, rather than a completely new way of interacting with the Web. A key difference between widgets and browser-based applications is that widgets are able to *interact with the platform* outside of a protected security sandbox. This has allowed widgets to gradually take the place of single-purpose applications on a user's desktop, further bluring the boundary between local and Web-based applications [21].

Widgets are relatively easy to develop and various implementers have targeted the large pool of Web developers to develop for their platforms [21]. This has resulted in a rapid rise and adoption of widget-based approaches and a number of implementations and definitions for what constitutes a widget and how it is developed and used. Different implementations have taken different approaches, and

so, many incompatibilities exist in the widget landscape. Widgets challenge browser-based approaches and so a number of questions exist regarding the suitability of widgets as RIA clients. Some of the key concerns of widget-based approaches include: the ease of development, maintainability, testing, portability, security, presentation and behaviour.

1.2 Research Report

The purpose of the research presented in this report is to critically evaluate the suitability and role of widget-based approaches. This has been done through an evaluation of the widget landscape and the creation of a conceptual widget framework, based on general characteristics and concerns. This framework has been used to develop a widget-based service so that advantages and limitations of widget approaches can be identified and evaluated. The contents of this research report are presented as follows:

Chapter 2: provides a literature survey of widgets and a definition for what constitutes a widget within the context of this research. Incompatibilities in the widget landscape and standardisation efforts being carried out to address these, are discussed.

Chapter 3: presents the research questions being addressed, expected outcomes and the methodology followed.

Chapter 4: describes the general characteristics and concerns of widget technologies and the conceptual cross-platform architectural framework developed.

Chapter 5: describes the requirements, specifications and design of the widget-based service developed.

Chapter 6: describes the implementation details of the service developed.

Chapter 7: presents the findings based on the service developed.

Chapter 8: summarises the research report and findings.

Chapter 2

Survey of Literature

This chapter provides a definition of what constitutes a widget within the context of this research and an overview of widget runtime environments on different platforms. The state of widget technologies is discussed with reference to incompatibilities in the landscape and standardisation efforts being carried out.

2.1 What is a Widget

2.1.1 Definition

Jaokar et al. define a widget as "a downloadable, interactive software object that provides a single service such as a map, news feed ..." [19]. Widgets however, are not restricted to providing a single service, and an application such as a map widget could also provide the weather for a user's current location. A number of widget technologies are actively under development and so numerous definitions exist for different implementations. The definition used for this research is based on the W3C's widget family of specifications, which describe a widget as:

> A *packaged*, *interactive client-side application* developed using *Web standards* and techniques [22].

All the resources defining the appearance and behaviour of a widget are bundled for distribution in a single package. Unlike a Web page, a widget is hosted on a client and able to access data services both on the Web and host device. For the

purposes of the research conducted, only widgets authored using Web standards are evaluated.

2.1.2 Alternative Terms and Definitions

Alternative terms for widgets include *gadgets, badges or modules* [4]. These alternative terms are evident in implementations such as Google Desktop or Windows Live Gadgets. The characteristics of a gadget are essentially the same as a widget, although the more common term widget is used throughout this report.

An alternative usage of the term widget is for a *widget toolkit*. This is a collection of elements forming part of a Graphical User Interface (GUI) framework as described in [23, 24] and does not fall within the context of this report.

Web-Widgets

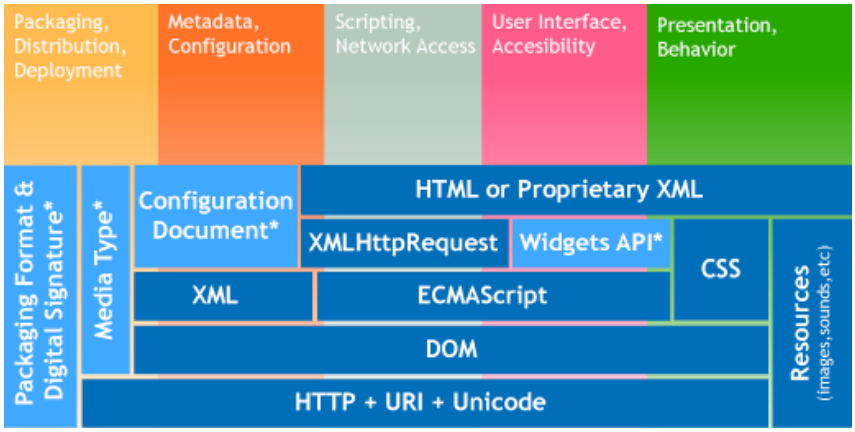
The term widget is often used to describe both a widget hosted on a client platform and a widget embedded in a Web document being served to a Web browser. Widgets embedded in a Web document are defined here as *Web-widgets* as they differ from the widgets being considered. A Web-widget consist of fragments of HTML, CSS and JavaScript that is dynamically or declaratively included in another Web document prior to it being served to a client [25]. Common examples are the gadgets available for the iGoogle homepage [26]. As widgets are authored using many of the same technologies and techniques as Web-widgets, a confusion between the two forms often arises. Functional similarities exist in how the applications are authored and behave, however, Web-widgets run within a browser and are restricted from directly accessing the host platform. Further differences include the implementation of internationalisation, localisation, security, packaging and APIs [25]. For this reason, only widgets hosted and instantiated on a client platform are being evaluated.

2.1.3 The Widget Engine

A widget is not a compiled binary application, but rather a package containing all the resources required to define the appearance and behaviour of an application. Widgets require a *widget engine* to render the user interface, handle user interaction and provide a programmatic means of accessing data services both on the Web and

host platform. The widget engine is a runtime environment that serves as a layer, decoupling a widget from the host platform. The engine provides widgets with interfaces to the underlying platform and is responsible for instantiating a widget as well as handling all behaviour [21, 27, 28].

In many aspects, widget engines serve a similar role to that of Web browsers, and many are built directly on top of Web browser frameworks to incorporate functionality such as: rendering markup and interpreting client-side scripting. An interpretation of the technology stack of a widget engine, as introduced in the W3C widget landscape document, is illustrated in figure 2.1 [25]. This stack does not represent any particular engine and is used to illustrate the underlying Web standards that widget engines should support and potential areas requiring standardisation. Standardisation issues are discussed further in section 2.3.2.



*Areas identified that require standardisation [25]

Figure 2.1: W3C widget engine technology stack

2.2 Widget Platforms

The ability to provide converged services is a result of improvements in the processing power and data capabilities of network enabled devices [29]. The prevailing platform supporting widgets is the personal computer, but widgets are being supported on a range of service delivery platforms that include the mobile phone and television. Many features of widgets are commercially and strategically desirable and so there has been substantial attention directed towards the use of widgets to deliver RIAs on a wide range of platforms.

The three main approaches for delivering RIAs on a particular platform include:

1. Developing and compiling a Web-enabled application for a platform's native operating system, such as a Symbian Mobile or iPhone application [30, 31].
2. Developing a Web application compatible with a Web browser on the platform, such as the Opera mobile browser [32].
3. Developing a Web-enabled application for a runtime environment supported by the platform, such as a widget engine or Java virtual machine (VM) [33].

The development of applications for a native operating system provides the most flexible and powerful way of building an application for a particular platform (assuming a development environment is available). The trade-off is the challenge of porting and maintaining an application for different operating systems and platforms.

Web browser-based approaches support the greatest number of platforms and allow for centralised maintenance and distribution. The main disadvantage of using Web browsers for running client-side applications is their limited access to underlying platform capabilities and incompatibilities between different browser's scripting and rendering capabilities [3].

Runtime environments may not always provide the most flexible or efficient approach, but they help to significantly reduce the complexities of cross-platform portability and development [33]. This is done by the provision of consistent programming interfaces to high-level language virtual machines [33, 34].

Widgets follow a runtime environment approach to delivering RIAs on a particular platform. A widget engine serves a similar role to a virtual machine, but unlike many established virtual machines (such as the Java VM), there are numerous incompatible widget engines for different platforms. Some of the prominent widget engines are listed in table 2.1. The Opera widget runtime is Java based and so runs on any platform that hosts a supported Java virtual machine [35].

2.2.1 Mobile Widgets

Various attempts have been made to bring Web applications and content onto mobile phones. As discussed by Reynolds [28], in early attempts, where memory and computational power were limited, the focus was on simplification. This is

Table 2.1: Prominent widget engines

Platform	Widget Engine	Operating System
Desktop	Konfabulator	Windows, OS X
	Google Desktop	Windows, OS X, Linux
	Windows Live Gadgets	Windows
	KDE Plasma	Windows, OS X, Linux
	Dashboard	OS X
Television	Yahoo! Widget Channel	Intel Multimedia Platform (Linux based)
Mobile	Web-Runtime	Nokia S60
	Vodafone 360	Mobile phones installed with custom Opera runtime
Cross-platform	Opera Widget Runtime	Linux, OS X, S60, Solaris, Windows, Windows Mobile

evident in efforts such as Compact HTML (CHTML) and the Wireless Application Protocol (WAP)[28]. Two reasons why early attempts failed are that [28]:

1. Content providers did not want to author multiple versions of the same content.
2. Users found the mobile Web browsers restrictive when compared to desktop browsers.

Mobile Web browsers have improved significantly but are still restricted from accessing platform capabilities. It is not uncommon to have a phone with one or two cameras, bluetooth, GPS, tilt-sensors and permanent Internet access. The relatively recent introduction of Apple's iPhone and Google's Android mobile operating systems [31, 36], have raised the competition for providing better development tools for mobile platforms [28]. New tools are encouraging the development of mobile applications that make extensive use of both services available on the Web and host device. As demonstrated by Apple's iPhone Application Store, from April 2009 more than one billion iPhone applications were downloaded over nine months [37]. Although the iPhone represents a small percentage of mobile users, it has shown the significant demand from both developers and consumers to access device capabilities that integrate well with Web services outside of a conventional Web browser interface [38]. As Apple's platform is proprietary and closed, applications cannot be shared across different platforms.

The market for mobile device applications is extremely fragmented and developers must choose between several incompatible application platforms that include Java, Symbian, Windows Mobile, iPhone, Android, and Linux [28]. The Java Micro Edition virtual machine (J2ME) is a popular runtime environment available on many devices. J2ME specifications are designed to expose many vendor-specific APIs and so Java is not fully supported by all mobile devices [28].

Mobile widget platforms have started to gain substantial support from operators and handset manufacturers [28, 39]. In April 2007, Nokia released the Web Runtime, a widget engine for the Symbian Mobile operating system to allow developers to create mobile applications based on HTML and JavaScript [39]. Since the introduction of the Web Runtime engine, a number of widget engines have subsequently been introduced by Windows Mobile, BlackBerry and Vodafone [40–42].

2.2.2 Television Widgets

Interactive Television (ITV) is a form of television where the viewer is capable of interacting with media based on their personal choices and physical interactions. ITV has a long and extensive history of approaches to create interactive experiences on television platforms. As discussed by Jensen [43], the evolution of ITV over the last 50 years is characterised by distinct phases plagued by a number of failures. Failures have resulted from a number of reasons, which include the use of premature technologies, expensive infrastructures and poor consumer adoption [43, 44].

The Internet and Web has fundamentally changed how consumers interact with multimedia and services. In contrast to the major failures experienced by ITV, the Internet and Web has experienced rapid growth and disrupted previous attempts to deliver ITV. A more recent attempt to re-launch ITV, has been the convergence of broadcast services with services available on the Internet [45]. Early attempts include, providing users with services normally accessed on a personal computer, such as email, search and chat. Simply recreating an emulated desktop environment on television adds limited value to users, and so new strategies are required to take advantage of the vast capabilities of the Internet and Web [43].

A dominant approach to providing Web-based services on television has not yet been established. In recognition of the poor adoption of Web-based services on television platforms, a prominent development that has emerged is to use widgets. A noteworthy effort is the *Widget Channel* being developed by Intel and Yahoo!

[46]. The Widget Channel is a software framework that allows widget applications to run on Intel consumer electronics multimedia platforms. The framework may be included on television sets or set-top-boxes by original equipment manufacturers and it aims to provide seamless Web experiences on television.

The Widget Channel Software Development Kit (SDK) has been made available to registered third party developers as of July 2009 [47]. This is not unusual in Web (and more recently telecommunications) communities, but it is in contrast to the very closed and proprietary television and set-top-box communities [48]. It is impossible for a single service provider to supply sufficient applications to cater for all niche markets and user preferences, and so the approach taken in launching the Widget Channel shows a more open strategy.

2.3 The State of Widget Technologies

The widespread adoption of widgets and widget engines has raised a number of issues for vendors, developers, users and new entrants. Widget-based technologies are still undergoing development and substantial fragmentation exists between different implementations. A number of efforts are being carried out to improve widget technologies and address the incompatibilities in the widget landscape. An overview of the state of widget technologies and standardisation efforts is provided, as discussed in [13].

2.3.1 Incompatibilities

Widget engines on different platforms face similar challenges and provide comparable implementations. The rapid evolution of widgets, has however resulted in a number of incompatible engines. These incompatibilities are a natural consequence of any new market, where vendors compete by creating new products with new features and services to differentiate their offerings [49]. As described by Tushman et al. [50], new technologies undergo an era of ferment in the technological life cycle, prior to the emergence of a dominant design which may or may not become a standard. A dominant design for RIA clients (or widgets) is yet to emerge, as a number of competing approaches continue to exist.

Kaar demonstrates cross-platform incompatibilities by attempting to port a Really

Simply Syndication (RSS) widget developed for Apple Dashboard to the Nokia S60 platform [27]. It is shown that significant modification is required to port a widget resource between two prominent engines. Incompatibilities encountered in the investigation include:

- Incompatible engine APIs.
- Dependencies to platform specific binaries and multimedia files.
- Incompatible configuration files and file systems.

Many widget implementations ignore or have yet to resolve certain technical issues relating to: development, packaging, configuration, metadata, internationalisation, maintenance and security. A significant limitation of proprietary widgets is that a user cannot run a widget developed for one widget engine on another widget engine without significant modification to either the widget or widget engine [27]. The W3C widget landscape document evaluates and compares a selection of mainstream widget engines to identify incompatibilities and areas of fragmentation [25]. Engines evaluated include: Yahoo! Konfabulator, Windows Vista Sidebar, Google Desktop, Opera Widgets, Apple Dashboard and the S60 Web-Runtime. Some of the key findings are summarised here.

Development

The development approach to author widgets is similar across different engines. Widgets differ from compiled binary applications in that they are authored using Web technologies. Widgets do not require any particular development environment or compiler as most can be created using any text editor and archiving application. All engines support common graphical resources (PNG, GIF, JPEG), scripting behaviour (usually JavaScript) and the `XMLHttpRequest` object to allow for Ajax requests. With the exception of Google Desktop, all engines support HTML and CSS for the user interface layout.

The area of greatest incompatibility in developing widgets for different engines is the difference between the APIs that allow widgets to perform common tasks and interact with a host platform. Common tasks include the handling of events, parsing strings, handling errors and accessing widget metadata. Examples of implementation differences include: how widgets manipulate user interface elements, open URIs, store persistent data and interact with host platform applications.

Packaging and Distribution

A key characteristic of a widget is that it is a single package, which includes any markup, styles, client-side scripting behaviour and multimedia resources. A common approach used by existing engines is to package a widget resource in a Zip archive. Incompatibilities in the packaging conventions include inconsistent:

- File extensions.
- Internet media types.
- Zip specifications.
- Packaging structure.

Examples of incompatible implementation details for widget extensions and media types are provided in Table 2.2 [25].

Table 2.2: File extension and media type incompatibilities

Widget Engine	Extension	Media Type
Google Desktop	.gg	app/gg
Konfabulator	.widget	application/vnd.yahoo.widget
Web-Runtime	.wgz	application/x-nokia-widgets

Configuration and Metadata

Widget packages usually include a configuration file, which holds metadata about that widget (such as the author and description) and configuration parameters (such as start-up behaviour, required resources and physical dimensions). All engines evaluated use XML for their configuration files. Although the semantics captured are similar, there is a lack of consistent fields, namespaces and configuration parameters in the schemas used [25].

Internationalisation

Internationalisation allows a widget to operate in multiple languages without a need to significantly re-engineer the core application logic and structure. Mainstream

engines support a sub-directory based internationalisation strategy, where content and configuration files for different languages are placed in predefined directories [51]. Inconsistencies exist due to the different conventions used for the packaging structure and configuration files.

Maintenance Updates

Maintenance updates are required to upgrade distributed widget resources to new versions. There is a general lack of support for automated maintenance updates across different widget engines. Konfabulator implements updates by using a unique identifier and version number, so that it can check for new versions and allow a client to download and install a new version using the previous version's preferences [20].

Security Models and Digital Signatures

The role of a security model is to provide policies for what actions instantiated widgets are able to perform. Various incompatibilities exist for enforcing security policies, if they exist at all. Policies are generally relaxed compared to those of Web browsers. Widgets are typically able to read, write, modify and delete files; automatically upload and download files; execute local applications and perform cross-domain requests. While this allows for very powerful client-side RIAs to be authored, it presents a number of vulnerabilities and threats to users. Opera widgets follow a very tight security model adhering closer to that of a browser, while others are more relaxed [52].

Some widget engines support digital signatures to authenticate the integrity of a widget's contents from the time that it is signed. There is inconsistency as to how digital signatures are implemented and used by different engines and so this continues to remain an area of significant fragmentation [13, 25].

2.3.2 Standardisation Efforts

The areas of incompatibility highlighted in section 2.3.1 do not describe all of the incompatibilities across the widget landscape, but provide an overview of some of the issues faced by developers, distributors, engine vendors and users. As widget technologies continue to evolve and different engines introduce new features, it

is natural that further incompatibilities will emerge. Incompatibilities between implementations have the potential of restricting widgets from becoming globally ubiquitous, and so, numerous efforts are being carried out in an attempt to standardise various aspects of widgets. The growing number of platforms looking to support widgets raise various questions related to how widgets should be implemented, as well as what aspects may benefit from standardisation.

One of the main objectives of the W3C is to provide a vendor-neutral space for the development of recommendations, patent-free standards, technologies, and guidelines that serve the global marketplace. In doing so, the W3C hopes to "lead the Web to its full potential" and ensure that the Web remains a royalty free medium for delivering information in a uniform way, to benefit any user on any computing system [53].

In 2005, the W3C identified widget-like technologies, which they initially called *Rich Web Clients*, as relevant to the future use of the Web and subsequently formed the Web Applications (WebApps) Working Group. The WebApps group hopes to improve client-side application development on the Web by providing specifications for markup and APIs for controlling client-side application behaviour [54].

Process

The process being followed by the W3C to standardise widgets is illustrated in figure 2.2 and consists of a landscape analysis, a requirements specification and various functional specifications and test suites. All tasks feed back on each other as the landscape is constantly changing and responding to disruptive innovations and new requirements from working group members. The standardisation process operates under a consensus model, where agreement is reached through collaboration between implementers, developers and other consortia. Collaboration is carried out using the W3C's public mailing lists and industry funded events and workshops [55].

The purpose of the landscape analyses conducted by the WebApps group is to identify incompatibilities between widget implementations and areas that may benefit from standardisation [25]. The task of requirements gathering involves actively working with various stakeholders to formalise mandatory and optional features that should be specified. These are formally outlined in the W3C's draft requirements document [22] and specifications for these requirements form part of the W3C's *Widget Family of Specifications*:

- Widget Packaging & Configuration [51]: Packaging, configuration data and internationalisation conventions.
- Widgets 1.0: Widget URIs [56]: URI scheme to address resources within a widget package.
- Widgets 1.0: Digital Signatures [57]: Concerned with defining the integrity of a widget package after it is created.
- The Widget Interface [58]: APIs to access features such as metadata and persistent storage.
- Widgets 1.0: Access Requests Policy (WARP) [59]: Security model to define network access.
- Widgets 1.0: Updates [60]: Maintenance update process.
- Widgets 1.0: View Modes Media Feature [61]: Concerned with presentation modes.

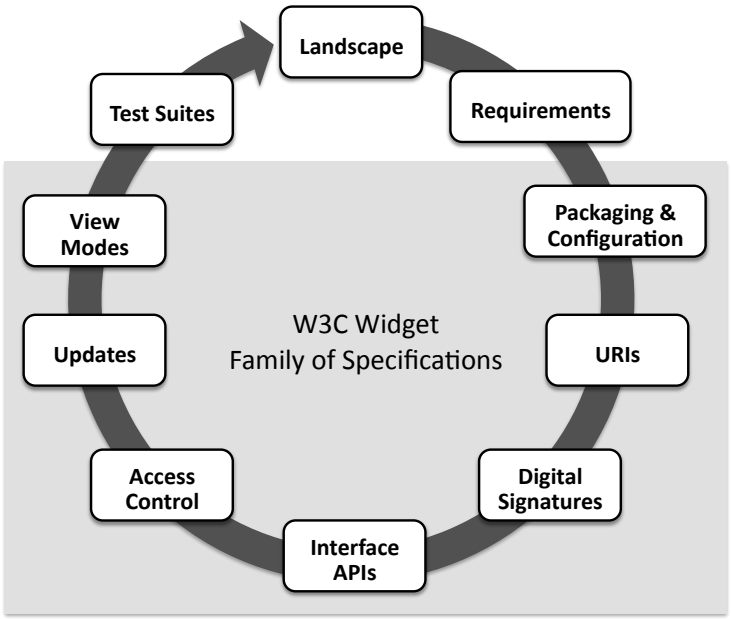


Figure 2.2: Overview of the W3C widget standardisation process

Complementary Efforts

Restrictions imposed by the WebApps Working Group Charter, limit what aspects of widgets can be standardised by the working group [54]. The W3C's standardisation

efforts explicitly lack a detailed security model, integration with Web documents or any APIs to access device capabilities. To address issues outside of the scope of the W3C specifications, a number of complimentary efforts are being undertaken by various consortia (particularly for mobile devices) and include:

- The Open Mobile Terminal Platform's (OMTP) *Bondi* initiative [62].
- The Join Innovation Lab's (JIL) widget platform [63].
- The Open Ajax Alliance's Gadgets and API Task Force [64].

OMTP is a mobile operators and manufactures consortium with nearly 40 participating companies that include many influential telecommunications companies: AT&T, Vodafone, Nokia and Sony-Ericsson. OMTP's Bondi initiative is focusing on defining a security policy language for widgets, and APIs to access device services such as the ability to send an SMS, make a phone call, take a picture and access a media gallery [62]. Bondi defines eleven APIs and relies on the W3C's Widget Family of Specifications for all other general functionality.

JIL is a collaboration between prominent mobile operators Vodafone, ChinaMobile, Verizone and SoftBank [63]. JIL is focused on creating a single widget engine for different mobile devices and hopes to encourage developers to create mobile widgets with a reach of "approximately one billion customers" [65]. JIL's activities are closed and proprietary, although beta implementations show evidence that JIL is using Bondi and W3C specifications.

The Open Ajax Alliance is an open consortium of companies, focused on standardising various development aspects of the Ajax development methodology and tools. Some prominent members of the alliance include Microsoft, Adobe, IBM and the Mozilla foundation. The Open Ajax Alliance's Gadgets and API Working Groups are primarily working on standardising Web-widgets deployed on the server-side [64].

There is currently no equivalent effort for standardising widgets on television platforms as there are for desktop and mobile widgets.

Benefits

As noted by [19], "Widgets harmonise applications development across the Web and enable a wider application distribution model". Standardisation has some of the

following potential benefits for the widget market:

- It will make it easier for new implementers to build standards-based widget engines from complete and open specifications.
- Developers will have access to larger consumer markets if they build widgets to conform to standard platforms.
- Developers will not have to support multiple versions of the same application across different platforms.
- Users will not have to install different widget engines.

2.3.3 State of Widget Standardisation

Significant importance has been placed on creating standards-based widget implementations from common specifications. This is evident by the collaborative contributions and adoption of specifications by prominent widget implementers and organisations that include Microsoft, Nokia, JIL, OMTP and Opera [22, 66].

The state of widget standardisation is largely dependent on the W3C specifications. Widget specifications are still in a relatively immature state as many of core specification documents are still in draft form. As outlined by the W3C Process document [55], specifications reach the following levels of maturity prior to becoming a W3C recommendation:

1. FPWD: = First Public Working Draft
2. WD = Working Draft
3. LCWD = Last Call Working Draft
4. CR = Candidate Recommendation
5. PR = Proposed Recommendation
6. REC = W3C Recommendation

Once a specification has reached W3C recommendation status, it is still not considered to be a standard, but may become a standard in future. The maturity of the specifications are summarised in table 2.3. As packaging does not generally

present any significant competitive advantage between implementations, it has been identified by the W3C as a priority concern that will benefit from standardisation. These specifications have been addressed prior to others and so are at a more mature level. Automated test suites have started to be developed to validate implementations against the W3C specifications [59, 67].

Table 2.3: Maturity of W3C widget family of specifications

Specification	**State**
Widget Packaging & Configuration	CR
Widgets 1.0: Widget URIs	LCWD
Widgets 1.0: Digital Signatures	CR
The Widget Interface	CR
Widgets 1.0: Access Requests Policy (WARP)	LCWD
Widgets 1.0: Updates	FPWD
Widgets 1.0: View Modes Media Feature	FPWD

Chapter 3

Research Question

Widgets present an opportunity to formalise the approaches that are used to develop and deploy RIAs for multiple platforms. The popularity and rapid rise of widget-based approaches across multiple platforms has resulted in a loose definition for what constitutes a widget and how it is developed, deployed and used.

A key problem identified with widget development is that a standard framework for developing widget applications has not been established. Widget technologies are undergoing substantial development and so approaches used to develop widgets are not yet standardised. Not all aspects of widgets are suited to standardisation although, many common concerns exist which may benefit from standardisation. As discussed in section 2.3.1, addressing the fragmentation of widgets without restricting innovation is a challenging task that requires continued collaboration between all major stakeholders. Standard practices and improvements to widget technologies will naturally result from these activities.

A further concern of widgets is the suitability of widget-based approaches for developing, deploying and maintaining RIAs. The suitability of widgets requires the consideration of various requirements and concerns, such as the ease of development, security and maintainability. Following this, the core questions that this research report aims to address are:

> What are the requirements for developing and deploying RIAs using widget-based approaches?
>
> Do widgets successfully meet these requirements?

3.1 Expected Outcomes

The main objective of the research conducted is to evaluate the suitability of widgets when used for RIAs. The strengths and weaknesses of widgets are evaluated based on the core requirements and features listed in table 3.1.

Table 3.1: Widget requirements being evaluated

Requirement	**Feature**
1. Behaviour	The ability for a widget to interact with the platforms and any local or networked application.
2. Ease of Development	The ease of developing, maintaining and testing a widget.
3. Security	The ability to prevent malicious behaviour from occurring.
4. Portability	The ease of packaging and porting widgets between different platforms.
5. Presentation	The ability to present an interface to a user and handle interaction.

3.2 Methodology

The methodology used to answer the research questions proposed is as follows:

1. Perform a review of the widget landscape.
2. Identify and model general characteristics and concerns to develop a conceptual widget framework.
3. Design, build and test a widget based on this framework.
4. Evaluate the application developed and draw conclusions.

The first phase serves to review the widget landscape to define what constitutes a widget and RIA within the context of the research conducted. The role of widgets for RIAs and the state of widget technologies are evaluated. This review forms part of the literature survey presented in chapter 2.

Following the review, common requirements, characteristics and concerns of widget technologies are identified and used to develop a high-level conceptual widget framework. This framework is presented in chapter 4 and models the requirements for widget-based RIA implementations.

In the third phase a prototype multimedia widget has been designed, developed and tested for a television service using the framework developed. The focus of the research conducted is to use the service developed to evaluate the role and suitability of a widget-based approach, and not to describe the implementation details required to support a widget on a television platform. The motivation for choosing to develop a multimedia widget is that multimedia services are not restricted to television platforms and may be offered on other platforms, such as a desktop, game console or mobile device. The design, development and testing of this service is presented in chapters 5 and 6.

In the final phase, the service developed is used to evaluate widget-based approaches by identifying and discussing issues related to the requirements listed in table 3.1. The findings and outcomes of the evaluation are presented in chapters 7 and 8.

Chapter 4

A Conceptual Widget Framework

As with any new technology, different implementations take different approaches to develop common features. An evaluation of the widget landscape and incompatibilities between widget engines as presented in chapter 2 and [13], has assisted in identifying general characteristics and concerns of widget-based technologies. A model of these concerns has been developed and used to describe a high-level conceptual framework for widgets.

4.1 Common Characteristics and Concerns

Perhaps a more general definition of a widget is that it is an updatable structured *package* containing *presentation* resources and *behaviour* logic capable of interacting with a set of APIs on the host platform in a *secure* way. A model of widget concerns and their relationships has been developed and is illustrated in figure 4.1. The purpose of this model is to provide an abstracted overview of core widget concerns and the relations of common characteristics. The four concerns identified are packaging, behaviour, security and presentation. These concerns form part of the technical requirements of a widget application.

4.1.1 Packaging

The conventions used to package a widget resource is a core concern for all widget implementations. With reference to figure 4.1, packaging encompasses issues related to how the structure should be implemented to archive all resources required by a widget. Resources within a widget package include configuration files, metadata,

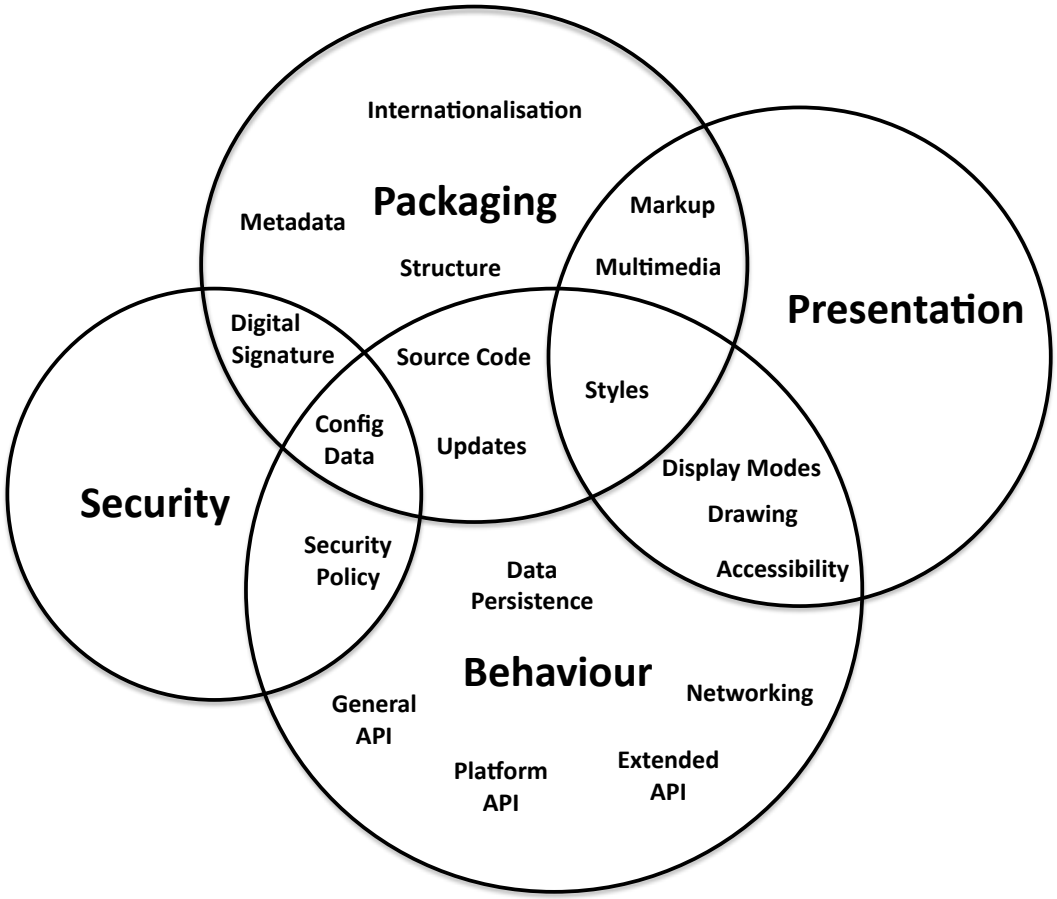


Figure 4.1: General characteristics and concerns of widget technologies

source code, markup, styles and multimedia [22]. The structure as described here, refers to the way paths to resources and configuration files are defined. Issues such as internationalisation, for example, rely on conventions to specify where resources for different languages are located.

The main motivation for packaging widget resources is to support the hosting of widget galleries (or application stores) and allow users to pass entire widgets between different platforms as single files. A package should be capable of supporting a digital signature to provide a widget engine with a means of validating the integrity of a package from the time that it was authored [22].

4.1.2 Behaviour

The behaviour of a widget resource as described here, refers to how a widget interacts with the host platform, users and any networked resources. The source code within a widget package interacts with the host platform through a set of APIs exposed by the widget engine. A widget engine is responsible for providing an interpreter to execute any client-side scripting code.

Behaviour concerns are currently facing the greatest incompatibilities between different engines as implementers are still developing new capabilities for widgets on their respective platforms. APIs to access device specific (and often proprietary) functionality are not always well suited to standardisation. Functionality, such as setting the recording preferences on a television or accessing the call history on a phone are very difficult to generalise without restricting innovation. A number of operations such as registering event listeners, storing persistent data and making HTTP requests are however, common to all widget implementations and so the potential to standardise certain behaviour concerns exists. To address this challenge, three kinds of APIs have been identified:

- *General API*, refers to functionality common to all widgets on all platforms such as: getting and setting user preferences, accessing persistent data, parsing strings and handling events such as closing a widget [58].
- *Platform API*, refers to functionality common to a particular platform. This may include acquiring GPS coordinates or sending an SMS from a mobile platform.
- *Extended API*, refers to additional functionality provided to extend the General and Platform APIs. The role of this is to allow for innovative features to be introduced, so that a platform is not restricted to the standardised functionality provided by the General and Platform APIs. As an example, a particular platform may create an extended API to allow widgets to use facial recognition capabilities.

Although a formal distinction between these three APIs is not made within the widget community, this has been done here to distinguish the different types of behaviours. Evidence of this distinction is apparent in the standardisation efforts being carried out for widget APIs. The W3C interface specification [58] describes only general behaviour concerns, such as persistent storage and the requirements

specification [22] identifies the need to allow widgets to bind to third-party APIs "that allow access to device-specific resources and services" such as those being defined by Bondi [62].

A further important behaviour concern of widgets is how maintenance tasks are performed to update a widget package to a new version while maintaining any saved preferences. As widgets are hosted on a client platform, any maintenance updates require that an updated version of a widget package is re-deployed on a client.

4.1.3 Security

Security concerns refer to how a widget engine ensures that any widget running on a platform behaves as intended without compromising or accessing a user's device or data in a malicious or unauthorised way [68]. To counteract the potential damage that may be caused by malicious applications on the Internet, Web browsers are specifically designed to run Web content and applications within a restricted sandbox [69]. Restrictions imposed on widgets are more lenient and widgets may be configured to perform cross-domain requests, use particular ports and protocols, access and manipulate host files and interact with hosted applications. As a result of this, a number of security measures are required.

The security model assumed in the framework is one of role-based access control (RBAC) as discussed in [7]. In this model, a widget represents a subject given rights to fulfill particular roles. The *configuration data* describes the permissions that a subject is granted to fulfill roles defined by a *security policy*. It is the widget author's responsibility to define the permissions in the configuration data and so a level of trust exists between the widget engine and widget author. The engine assumes that the permissions given to a particular widget are sufficient and the author assumes that the engine will correctly enforce the permissions defined. Based on these assumptions, a further third party may be required to validate the integrity of a widget's behaviour and ensure that the restrictions imposed by the author are correct. A third party may also be required to validate that the security policy of an engine is enforced correctly.

A widget's source code is included within the package, and so unlike a compiled binary file, it is relatively easy to access and alter a widget's behaviour by editing the source code and configuration files included within the package. A widget engine should be able to determine the integrity of a package through the use of a digital

signature that ensures that the package is not altered during its lifetime [22].

4.1.4 Presentation

Presentation concerns refer to how a widget is displayed through the use of markup, styles and any supporting multimedia files (such as videos or images). Widget engines typically make use of existing layout engines, such as WebKit which is used to render Web content for different applications on different platforms [70]. Presentation concerns include:

- Libraries to support common User Interface (UI) elements such as text areas and combo boxes.
- The ability to draw primitive shapes such as lines and rectangles.
- Different display modes for when a widget is in their different states, such as loading, docked or in full screen.

A further characteristic of widgets are accessibility capabilities to allow people with different disabilities, the ability to perceive, understand, navigate and interact with widgets. There is currently very little attention being directed towards addressing widget accessibility.

4.2 Widget Framework

Based on the general characteristics and concerns of widgets introduced in section 4.1, a framework has been developed to describe the core components of a widget architecture, independent of the service delivery platform. This framework is best described using a series of high-level architecture diagrams.

4.2.1 Architectural Overview

The architecture of widget solutions for all platforms follows the basic principles of a Service Oriented Architecture (SOA) as described in [10]. As illustrated in figure 4.2, a requester (service delivery platform) acquires a widget through a discovery

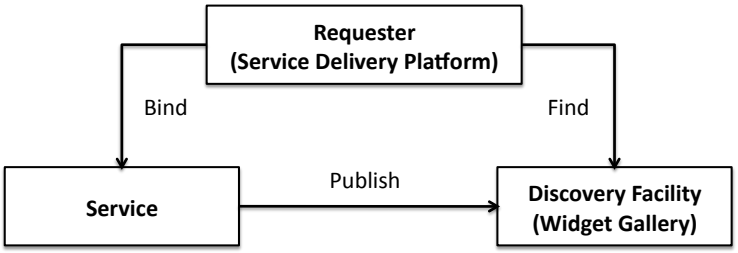


Figure 4.2: Widgets in relation to a service oriented architecture

facility (widget gallery) and then uses a widget to bind to a service to perform a particular task.

By extending this principal, a generalised conceptual architecture for widgets is illustrated in figure 4.3. Services provided to a service delivery platform may include a television broadcast or a telephony service delivered over an appropriate network. A particular network has not been specified here as different networks may be used for different services. If the platform is equipped with a widget engine and able to connect to a network, such as the Internet, then a typical scenario is that a user browses a widget gallery (or application store) to obtain widgets to add to their personal inventory. Once acquired, widgets are permanently hosted on a user's platform and may make use of a range of remote supporting services to perform a range of tasks. Services may include *primary services* made available to the platform through a principal operator, or *specialised services* made available through a third-party application provider.

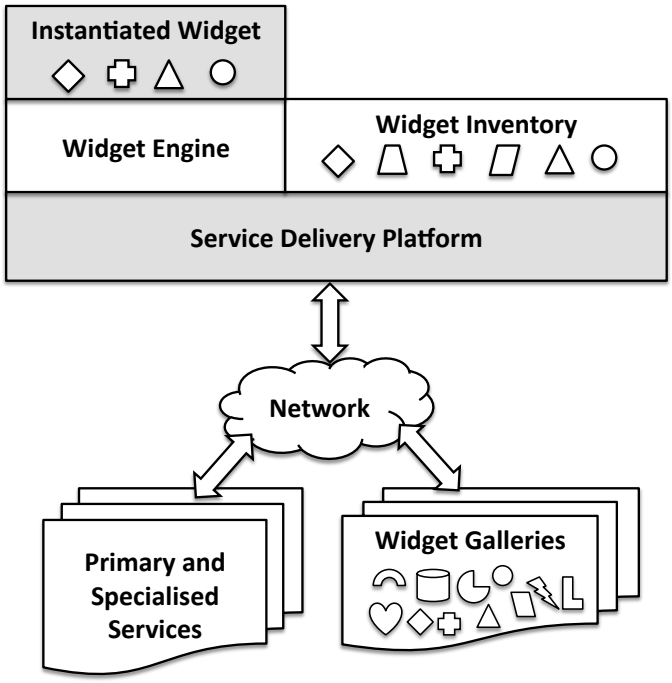
The architecture illustrated in figure 4.3 describes an ideal case where a widget may run on different platforms in a uniform way. Due to incompatibilities between widget engines on different platforms, this ideal case does not currently exist. For this case to be possible, widget engines must provide consistent interfaces to widget applications.

4.2.2 Widget Engine

A proposed architectural overview of a widget engine is illustrated in figure 4.4. This layered architecture serves to identify the functional units that a widget engine must provide to widget resources, based on the general characteristics and concerns identified. As widgets predominantly use JavaScript for client-side scripting, a JavaScript interpreter is required for handling behaviour. The widget engine is responsible for abstracting the complexities of underlying services and resources,

such as networking and data persistence from widgets. The general and platform APIs expose standard functionality and an extended API may expose specialised functionality. A native library hosted on the engine exposes these APIs. All tasks such as parsing strings or making Ajax requests is done using these APIs. The engine described in this framework is comparable to a Java virtual machine in that it provides a consistent interface to any application running on the framework and is responsible for hiding any differences between underlying operating systems [33].

(a) Instantiated widgets running on different service delivery platforms



(b) Architectural overview

Figure 4.3: Conceptual widget architecture

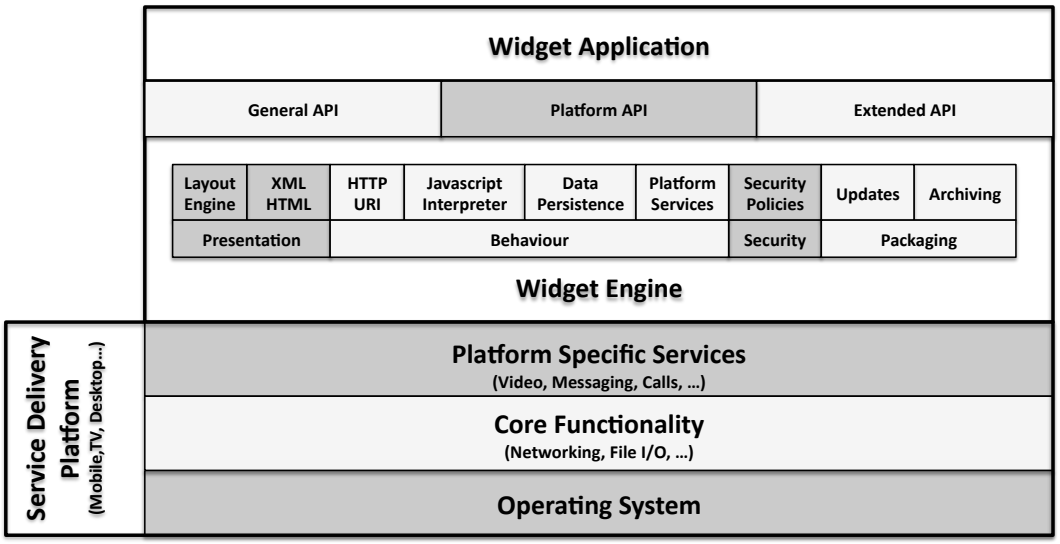


Figure 4.4: Conceptual widget engine architecture

Chapter 5

Specifications and Design

Chapter 4 provides a conceptual framework describing the basic architecture required to support widgets on multiple platforms. The approach taken to evaluate widget-based approaches is to use this framework to build and test a widget-based multimedia service for a single platform. A film rating and recommendation service has been developed that demonstrates considerable interaction between different problem spaces.

5.1 Problem Space

Multimedia services range in complexity from simple applications that notify a user when an event has occurred (such as a goal being scored), to complex applications allowing users to interact directly with video streams (such as finding out the name of an actor in a particular scene). Broadcast operators and content providers have recognised the importance of strategic alliances with specialist service providers, as it has become impossible for single entities to provide all the services required to satisfy the spectrum of expectations of modern entertainment systems such as ITV [45]. ITV's challenges are largely dependent on the evolution of next generation converged networks that allow users to interact with services hosted on different networks or domains [29]. The problem spaces being evaluated through the development of the recommendation service are that of the:

- Widget application.
- Widget engine.
- Television platform.

- Primary content service.
- Specialised application service.

5.1.1 Recommender Systems

The increasing volume of on-demand multimedia content has introduced a number of challenges for how users navigate and find relevant content, using interfaces with limited interaction [71]. As described by Koren et al [72], recommendation services assist users to find suitable products and have become core components of many e-commerce websites, such as Amazon and Netflix. Recommender systems are particularly suitable for entertainment products that include films, music, and TV series, as customers are willing to rate their level of satisfaction. The Web may be used to build extensive libraries of rating data that can be used to recommend appropriate content to consumers [72].

5.2 A Film Rating and Recommendation Service

Using the principles of a recommender system, a film rating and recommendation service, *FrameRate*, has been developed. The objectives of FrameRate are that it allows users to rate films that they are watching so that they are able to build a personalised history of films that they enjoy. Using this rating history, users are able to browse through recommended films for particular genres that they are likely to enjoy and have not watched. FrameRate provides a means of demonstrating:

- A user's relationship to films.
- A user's interaction with a television platform and widget.
- A widget's interaction with a television platform and primary content service.
- A widget's interaction with a specialised third party rating service.

5.2.1 Requirements

A set of requirements exist to identify current and potential future features of the service. The three core requirements identified for FrameRate are that it allows users to:

1. Rate a film during playback.
2. Browse recommended films for a particular genre.
3. Select and play a film on the host platform.

A user is not required to perform any specialist tasks and so the rating service is designed for limited user interaction. The complete requirements specification can be found in appendix A. Further requirements defined for the system have not been implemented, but have been used to identify desirable future features that should be considered in the current system design. These include: requirements for alternative interface modes, content sources, delivery platforms and smarter recommendation results.

5.2.2 Assumptions

Certain assumptions have been made to limit the scope of the rating service. Content being rated is restricted to a set of feature films. In practice, content could consist of a range of media including live broadcast, music videos, television series and user-generated content. It is assumed that video files are fixed and of a single quality for all clients. Clients are restricted to television platforms and widgets will operate in a single window mode overlaying a film. These assumptions significantly reduce the complexity of content provision concerns.

The focus of the FrameRate prototype implementation is to evaluate behaviour, security and presentation concerns for a single widget engine. Each widget engine uses different packaging conventions and so the ability to test different approaches to packaging widgets is not possible using a single engine. Packaging concerns have been extensively evaluated by the W3C and so fewer questions exist regarding a suitable packaging approach. The W3C's findings are provided in the widget packaging candidate recommendation [51].

5.2.3 Functional Specifications

The functional specifications of the rating service are based on a set of use cases. The use cases describe what the service needs to do to allow users to *rate*, *browse* and *play* a film on the platform. Functional specifications have been developed using techniques proposed by Cockburn [73] and the complete specifications can be found

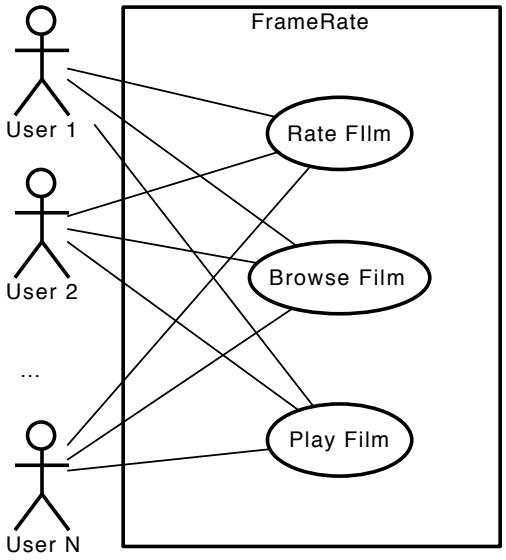


Figure 5.1: FrameRate use case diagram: general case

in appendix B. A use case diagram for a general case is illustrated in figure 5.1. A key outcome of the use cases developed is the role of a user as an actor on the system. In the IT, mobile and Web domains a user is normally characterised by a personal account or profile that may form part of a group of users. Television platforms differ from this in that interaction with the platform is not restricted to single users or entire groups. The use cases highlight that primary actors on the system may be a single user *or group of users* that may rate and browse films based on their *particular or collective* preferences. A group is dynamic in the sense that a user does not permanently belong to any group or action being performed. As an example, a group of friends may get together and collectively interact with the widget to browse for a film best suited to the group's interests. The interaction with the platform should not be restricted to a single user profile but rather based on the presence of multiple users.

5.3 Architecture

The design of the FrameRate architecture is based on the conceptual widget architecture illustrated in figure 4.3. The notion of a widget gallery and user inventory has been removed, as it is assumed that the widget is already available on the platform. The high-level FrameRate architecture is illustrated in figure 5.2 and separates primary and specialised services accessed by the delivery platform over an Internet Protocol (IP) network. FrameRate consists of three key subsystems:

1. Content Service (Primary Service)
2. Film Rating Web Service (Specialised Service)
3. Service Delivery Platforms (Clients)

5.3.1 Content Service

A content service is required to emulate a video-on-demand service. The responsibility of the service is to provide any platform connecting to it with on-demand access to a remote collection of films. Transport and storage implementation details are not a core concern of the rating service and so, only a simplified unicast streaming service is required to demonstrate clients accessing remote content.

5.3.2 Rating Web Service

A centralised Web service is responsible for storing and managing film ratings for all users on any platform. The Web service provides the computational intelligence required to generate film recommendations for any client. The service is designed to be standalone in that it is not restricted to widget clients and capable of serving any client able to use HTTP (such as a Web browser). The data model and functions exposed by this service are discussed further in section 6.2. This service is classified as a specialised service within the context of this work as it provides functionality to compliment the primary content provision service.

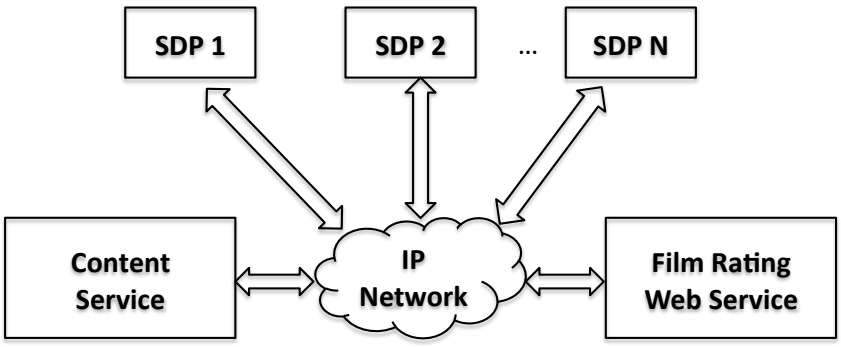


Figure 5.2: FrameRate high-level architecture

5.3.3 Service Delivery Platform

The service delivery platform hosts the media player, widget engine and widget application and is responsible for allowing a user to interact with content and rating services as illustrated in figure 5.3. The role of the widget is to provide a single interface for a user to interact with the platform and Web application in a seamless way. The widget serves as a front-end to the rating Web service, but is standalone, in that it does not need to be connected to the Web service in order for it to instantiate and run on the platform.

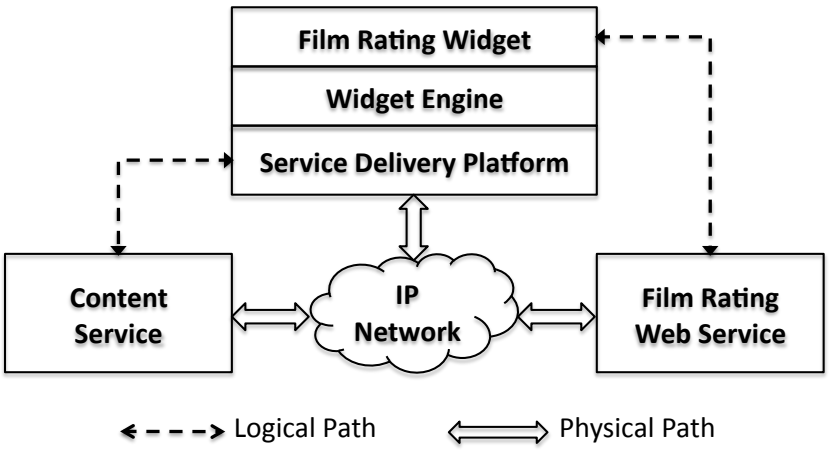


Figure 5.3: FrameRate architecture: single platform

Chapter 6

Development and Testing

The FrameRate service has been developed and deployed in the Convergence Laboratory at the University of the Witwatersrand's Centre for Telecommunications, Access and Services (CeTAS) [74]. The laboratory provides a configurable Next Generation IP Network (NGN) where the rating service can be deployed, configured and tested for different platforms. This chapter outlines the principal implementation and testing details of the content service, rating Web service and service delivery platform.

6.1 Content Service

The content service provides each delivery platform with access to a collection of films. Films are hosted on an Apache Web server and the HTTP protocol is used by platforms to request films on-demand. Apache is capable of independently streaming the same film to different clients. Each film is associated with a unique media identifier (`mid`), which is used to identify films requested from the Web server. HTTP is not an optimal protocol for controlling playback and video streaming, but it is sufficient to provide a simplified streaming service for test platforms to interact with remote content streams, as transport and storage concerns are not being considered. A more appropriate protocol for media streaming is the Real Time Streaming Protocol (RTSP) [75].

6.2 Rating Web Service

A number of approaches and technologies exist to develop and deploy the rating Web service. Key requirements of the service are that it is centralised, loosely coupled from the widget application and performs any computationally demanding tasks. A further implicit requirement is that message exchanges are simplified for the purposes of tracing, debugging and testing.

The Web service is responsible for capturing and storing all rating data and for generating intelligent recommendations based on a film and users' rating histories. The functional specifications identify the Web methods required to allow a client to rate a film for a number of users, get film meta-data and get recommendations for a particular genre. The widget invokes the service by making Remote Procedure Calls (RPC) [76]. The method name and parameters are passed within the URI of an HTTP request and results are returned as JavaScript Object Notation (JSON) encoded strings as demonstrated in listing 1. JSON is an alternative to XML that provides a suitable message exchange format for JavaScript clients. JSON strings represent JavaScript objects and so do not need to be deserialised[1] [77]. The Web methods exposed by the service include:

`bool TestOnline()`
: Test method to check if the service is available.

`json GetFilm(mid)`
: Returns a JSON encoded representation of a film.

`bool RateFilm(user_ids, mid, rating)`
: Rates a film for one or more users.

`json GetGenreFilms(user_ids,genre,rated)`
: Returns a JSON encoded ordered list of recommended films for one or more users. Films are returned for a particular genre and may or may not have already been rated before.

The approach used offers substantial flexibility, as it does not require clients to bind to the service. It also allows the service to be tested and used from other

[1] Deserialisation (or unmarshalling) as described here is the process of converting a serialised string representation of an object into an object that can be accessed and manipulated in the client code.

Listing 1 JSON response for single user film request

```
HTTP GET
http://frame-rate.appspot.com/rpc?action=
GetGenreFilms&id=testu1&genre=comedy&rated=false

HTTP/1.1 200
{"films" : [
  {"mid": "NapoleonDynamite",
    "title": "Napolean Dynamite",
    "description": "...",
    "genre": "comedy",
    "average_rating": "3.8"
  },{"mid": ...},{...},{...}
]}
```

clients, such as a Web browser. The use of more formalised Web service techniques (such as SOAP [10]) may offer a more robust and secure approach, but the increase in complexity does not provide any substantial benefit within the context of the investigation. Limited validation is performed and it is assumed that clients adhere to the correct conventions.

The rating Web service has been developed using *AppEngine*, a Google-based Web application server. AppEngine provides a framework to develop and deploy a Python based Web application capable of processing HTTP requests, managing persistent data and generating responses [78]. AppEngine is based on Google's virtualised cloud infrastructure and so applications are easily scalable on demand. The Web services may be deployed either on the AppEngine cloud or a locally hosted server.

The main motivation for choosing AppEngine is to use the Google Datastore API. Within an AppEngine application, a persistent data object is defined by a single class and there is no need to define further persistence layers or connectors to a back end database. This significantly reduces the complexity of the service by abstracting underlying data storage concerns. A simplified representation of the data model used is illustrated in figure 6.1. A resulting outcome of developing the rating service is the need for a unique key to identify all film entities for both the rating and content services. This `mid` key correlates to the `mid` used by the content service to identify film streams.

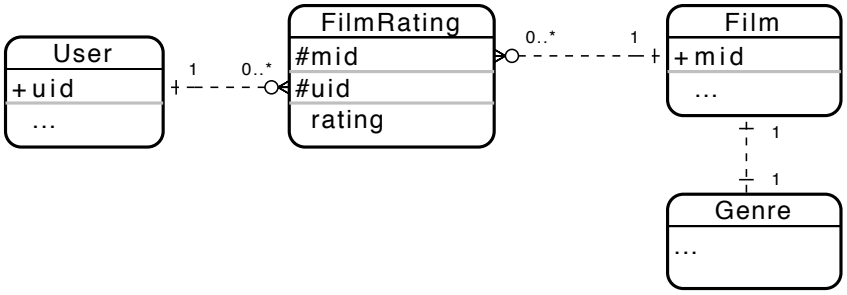


Figure 6.1: Simplified FrameRate Web service data model

For the purpose of the prototype, basic film meta-data is stored by the rating Web service. The widget acquires meta-data from the rating Web service as this allows the widget to obtain only relevant meta-data. It is preferable that the rating Web service is not responsible for hosting a meta-data library and that meta-data is either acquired dynamically from a third party (such as the Internet Movie Database (IMDB) [79]) or directly from the delivery platform through an appropriate content API. A disadvantage of storing meta-data on the rating Web service is that in future, the rating service becomes responsible for keeping its meta-data library up-to-date when new films are released. By delegating this responsibility to another service, the rating Web service does not need to be concerned with this additional responsibility.

6.3 Service Delivery Platform

The service delivery platform consists of the widget engine, media player and widget application. The implementation of these subsystems is briefly described.

6.3.1 Widget Engine

The service delivery platform is largely dependent on the choice of widget engine. As discussed in section 2.2.2, the prevailing widget engine available for developing widgets for television platforms is the Yahoo! Widget Channel for the Intel multimedia processor. The Widget Channel development kit may be used to develop, test and deploy widgets for supported television platforms. It has been decided not to use the Widget Channel implementation, as certain features (such as user profiles and interaction with media) are enforced by the Widget Channel implementation. The Widget Channel is based on the Yahoo! Konfabulator desktop engine and so, it has been decided to rather use the more generic Konfabulator engine, as it

provides a greater degree of flexibility to develop and investigate the interactions of the underlying subsystems [20]. The Konfabulator engine is currently only supported for the Windows and OS X operating systems and so a desktop environment is used to emulate the delivery platform.

The FrameRate service delivery platform architecture is illustrated in figure 6.2 and follows the conceptual widget engine architecture illustrated in figure 4.4. The native Konfabulator JavaScript API provides general functionality that includes timers, event handlers, manipulation of UI elements and access to the `XMLHttpRequest` object [20]. A custom platform API has been developed to expose media player functionality not supported by Konfabulator through a similar JavaScript interface.

6.3.2 Media Player

A customised instance of the VideoLan (VLC) media player is used to handle and process multimedia files. The VLC player natively supports playing HTTP video streams, by progressively downloading and buffering portions of a video file from the content service, to play films on the platform. VLC provides the ability to write macro scripts for the VLC HTTP interface to expose media properties (such as the `mid`) and player control [80].

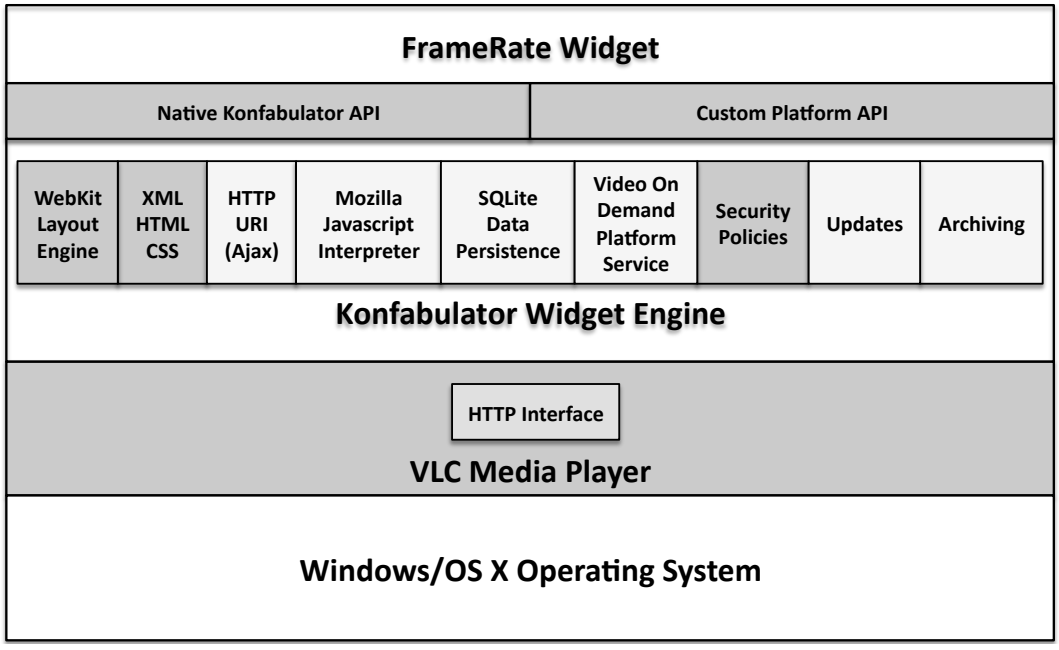


Figure 6.2: FrameRate service delivery platform architecture

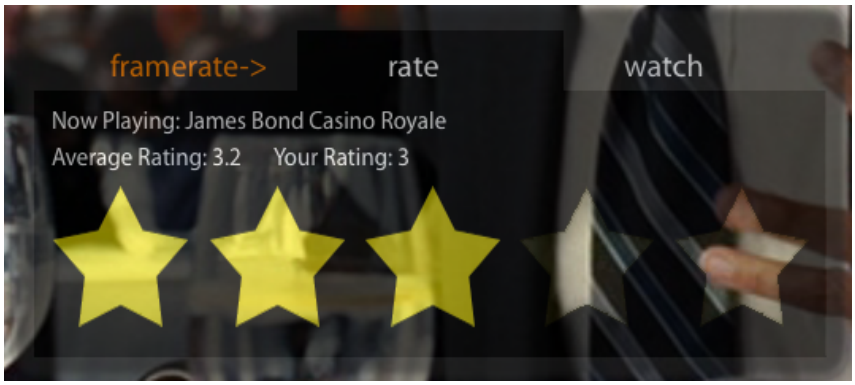
6.3.3 Widget Application

User Interface

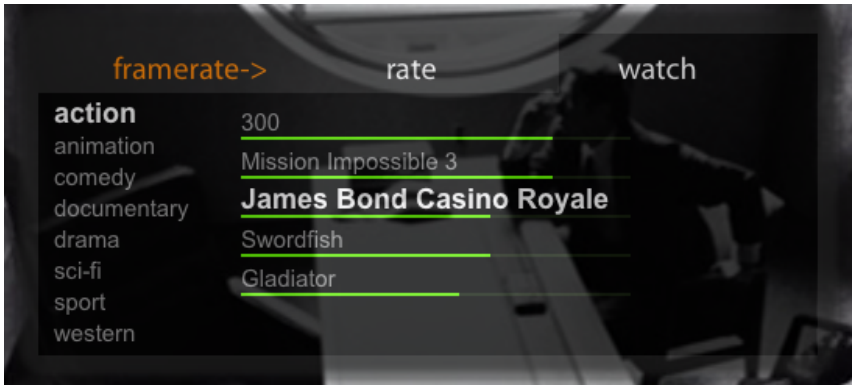
The widget UI is defined in XML and composed of layered Portable Network Graphic (PNG) images of varied opacities. Konfabulator uses the WebKit layout engine and it is configured to render the widget over the video streams as illustrated in figure 6.3. The Konfabulator implementation allows all scripting behaviour to be included in separate files and so the UI definition is well separated from all scripting and event handling behaviour.

Behaviour

The native and custom platform APIs are used to develop the FrameRate widget's behaviour. The native API provides timers to poll the platform to monitor any



(a) Rate mode for a rated film being watched



(b) Browse mode showing ranked recommendations

Figure 6.3: FrameRate widget screenshots

changes, event handlers to intercept and execute UI actions and the `XMLHttpRequest` object to perform Ajax calls to the Web service. As the UI definition is loaded on initialisation, the engine allows any element in the UI XML file to be accessed and manipulated directly, and so there is no need to navigate a DOM tree as is usually done for HTML based Web page applications.

The role of the custom platform API is to allow the widget to interact with media player functionality. The API does not form part of the widget package and is exposed by the engine. Konfabulator may interact with the underlying platform by:

- Making HTTP calls to the host platform.
- Executing shell commands on the host platform.
- Executing Applescript (OS X).
- Binding to a Component Object Model (COM) library (Windows).

The method chosen to develop the custom platform API is to make HTTP calls to the VLC HTTP interface. The custom API serves as a wrapper to the VLC HTTP interface that abstracts the FrameRate widget from the content service and media player. This signifies that detailed knowledge of the content service or VLC player is completely hidden from the widget. The widget is only aware of a `mid` and `TV` object when using the following JavaScript methods:

```
void TV.playFilm(mid)

mid TV.getTitle()
```

6.4 System Integration

The integration of the core subsystems is best illustrated by the message sequence diagram in figure 6.4. Only significant interactions are illustrated here for rating and browsing activities. In the approach used, the widget is responsible for managing its state by querying the platform and Web service for any state changes.

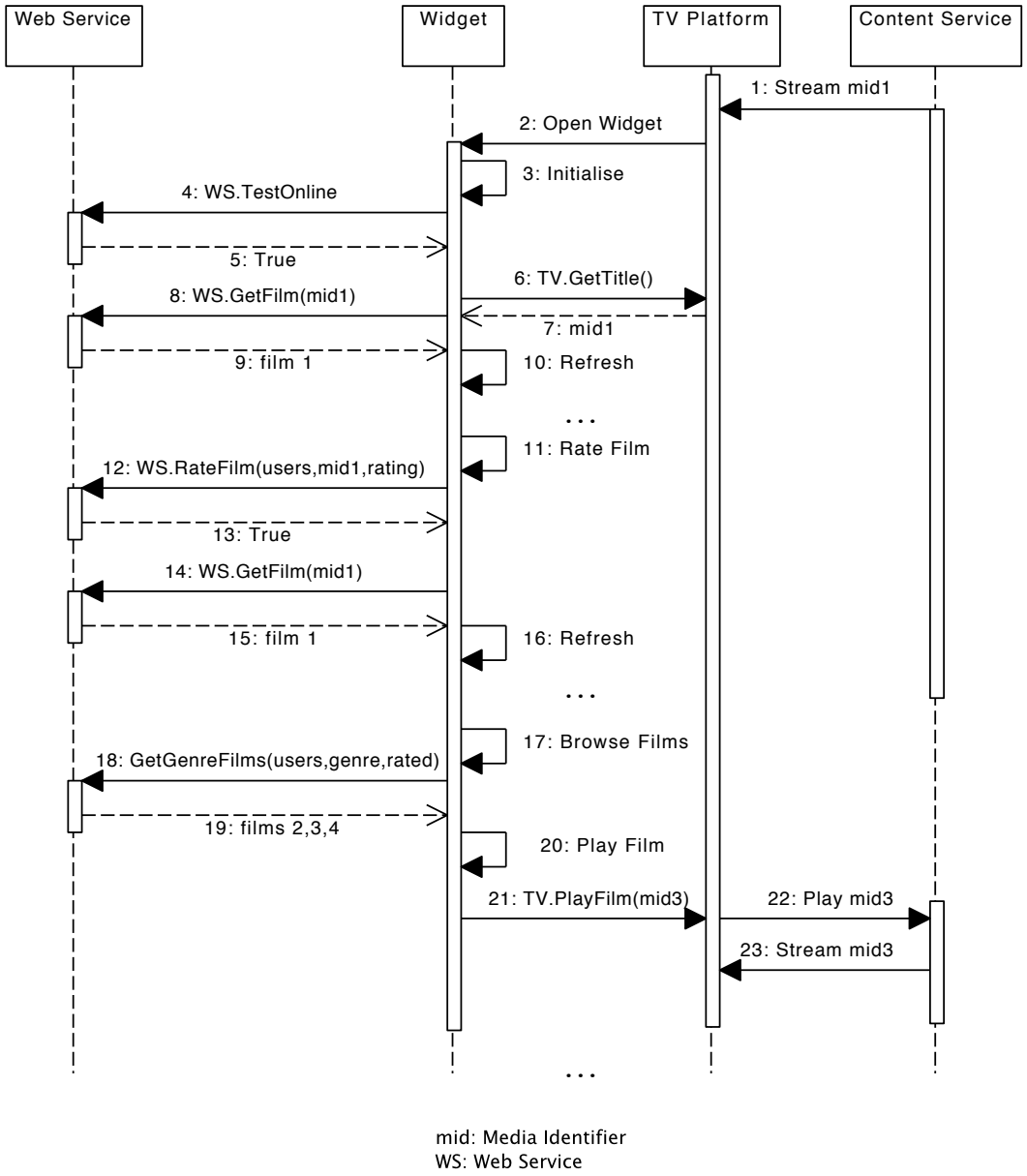


Figure 6.4: Message sequence diagram demonstrating integrated FrameRate service

6.5 Testing

No user trials of the service have been done and only system testing has been performed. The testing environment consists of two client platforms and a streaming content server hosted on the same network as illustrated in figure 6.5. The widget has been deployed on two client platforms running OS X and Windows respectively. As the content service is not central to the investigation, no formalised testing has been carried out for this service.

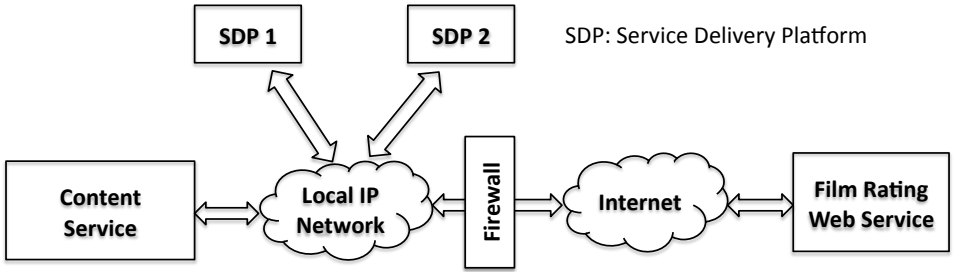


Figure 6.5: FrameRate service test environment

The Web service is deployed on the Google AppEngine cloud and test data has been generated for 35 feature films and 15 users. A test script is used to create the test users, films and rating profiles. A script is used to simulate and test typical activity flows of the Web service and to allow integrated regression testing to be performed.

Widget testing is largely event-driven and thus conducted manually by triggering events from the UI and simulating scenarios, such as the Web or content service going offline. Further work is required to evaluate the ability of performing automated regression testing within the Konfabulator environment.

Some of the integration tests conducted during the development of the FrameRate service include:

- User(s) rate(s) film that has (not) been rated.
- User(s) rate(s) film from different platforms.
- Widget Recovers from Web/Content service downtime.

6.6 Recommendations

The Intel Widget Channel implementation allows widgets to register event listeners for common events, which significantly reduces the passing of unnecessary messages between the widget and platform. A limitation of the FrameRate implementation is how the widget polls the platform for any changes in the content stream. A more suitable approach to the one used is to create a controller where event listeners may be registered for particular events (such as a channel or content stream changing).

Porting and testing widget clients on other platforms is technically possible, but due to incompatibilities between widget engines, this requires a complete rewrite of

the widget and platform API. It has been decided not to develop a widget for other platforms, even though this may assist in identifying alternative approaches used by other engines.

Chapter 7

Evaluation of Widget-Based Approaches

The conceptual widget framework developed in chapter 4 has assisted in identifying further features to the requirements introduced in section 3.1. The outcomes of the requirements and features evaluated, are summarised in table 7.1. Each feature is rated as either weak or strong, based on the investigation conducted. A discussion of the outcomes is presented in this chapter. As widgets use similar conventions and techniques to those used to develop Web browser RIAs, many of the findings are compared directly to browser-based approaches.

A number of the strengths and weaknesses of widgets have been identified through the development of FrameRate. Only issues directly influencing the service delivery platform are presented in this chapter and the supporting content service and film rating Web service are not evaluated further. Although FrameRate serves as a reference implementation, the findings presented are not specific to the Konfabulator widget engine or FrameRate application.

7.1 Behaviour

A widget engine is capable of providing widgets with access to a range of platform and remote application capabilities through simplified interfaces. The FrameRate widget interacts with both a remote Web service and platform media player. This is one of the main advantages of widgets, as simple applications may be created that are capable of using both platform and Web capabilities. Access to platform capabilities and different domains is usually restricted from Web browsers due

Table 7.1: Summary of requirements, features and outcomes

		Outcome	
Requirement	**Feature**	**Weak**	**Strong**
1. Behaviour	Ability to Interact with Remote and Local Platform Applications		X
2. Ease of Development	Abstraction		X
	Native Libraries		X
	JavaScript Development	X	
	Persistence and State		X
	Maintainability	X	
	Testing	X	
3. Security	Protect Against Threats	X	
4. Portability	Packaging		X
	Cross-Platform Portability	X	
5. Presentation	User Interface		X
	Styles	X	

to security concerns. Widget engines offer more flexibility to develop advanced applications, but as discussed in section 7.3, there are still a number of security concerns to consider.

7.2 Ease of Development

An objective of widget-based approaches is that they empower developers with the ability to use familiar Web techniques to create RIAs for a range of platforms [21]. Many of the advantages of widget-based approaches are as a result of abstracted interfaces that hide complexities and simplify RIA development.

7.2.1 Abstraction

The ability to abstract, encapsulate and expose behaviour within consistent interfaces, significantly reduces the complexity of a widget's source code. Although only a small subset of platform capabilities have been implemented for FrameRate, the custom platform API demonstrates the possibility and advantage of abstracting

platform complexities. The custom platform API uses the same conventions as the native Konfabulator API and so there is no difference between the programming conventions required to access both native and custom capabilities. The reusable `TV` object (introduced in section 6.3.3), does not need to be instantiated and may be used by any widget hosted on the platform to interact with the media player. A widget does not require specialised knowledge of the platform and is completely unaware of how content is transported and played. The ability to use simple interfaces that expose local and remote capabilities is a desirable feature of widgets, and one of the principal strengths of widget-based approaches.

7.2.2 Native Libraries

Widget-based approaches have been found to simplify many aspects of RIA development, especially when compared to browser-based approaches. Developing RIAs for Web browsers usually requires the inclusion of supplementary JavaScript libraries that support more advanced features such as handling Ajax, creating and manipulating UI elements and parsing strings. JavaScript libraries for Web browsers have gained considerable popularity as they help to simplify RIA development and assist in masking the many incompatibilities and challenges of writing JavaScript for different Web browsers [11, 14]. Some of the popular libraries include Dojo and JQuery [81, 82] and many other libraries are available to provide specialised capabilities such as maps and social networking [14]. Due to the flexibility provided by the JavaScript language, the conventions to use these libraries are inconsistent, and the choice of library is often based on a developer's personal preference and experience.

Konfabulator's native JavaScript API includes functionality that is normally accessible in Web browser development through the use of additional libraries and plugins. Konfabulator's native API significantly improves the intuitiveness and consistency of techniques required to use more advanced JavaScript capabilities. As there is less of a distinction between the conventions of the native APIs and that of additional libraries, RIA development is significantly simplified for widgets when compared to browser-based approaches.

7.2.3 JavaScript Development

The popularity of the JavaScript language can largely be attributed to the extensive adoption of Ajax programming techniques [14, 28]. JavaScript has rapidly evolved from a simple scripting language used to enhance the interactivity of Web pages, to an advanced language capable of being used to develop elaborate client-side applications and interfaces. Although less common, JavaScript may even be used on the server-side [83].

JavaScript is a dynamic, weakly typed scripting language that allows new functions to be defined and altered during runtime [83]. As a result of this flexibility, it is very difficult for development environments to detect coding errors prior to runtime, and debugging capabilities are limited [14]. The extensive usage of JavaScript has promoted the improvement of development environments. Frameworks such as the Google Web toolkit have been developed to provide more structured development environments where RIAs may be written in Java and compiled to JavaScript [16]. A number of further efforts are being carried out to optimise the efficiency of JavaScript interpreters through the development of just-in-time compilers [84].

The large pool of JavaScript-aware developers may have created a perception that JavaScript is a relatively easy language to learn and use [85]. While the use of JavaScript may allow more developers to create widgets, the large base of developers itself does not ensure that JavaScript is the optimal or easiest language to use for developing RIAs. Due to the immense flexibility and inefficiencies of JavaScript, it is argued that, although it is technically possible, the usage of JavaScript is not optimal for building larger more complicated RIAs.

7.2.4 Persistence and State

A Web browser's primary objective is to allow users to navigate between different Web pages. One of the consequences of Ajax based approaches is a difficulty in maintaining a client's state as a user navigates between pages [11]. This is a significant strength of a widget-based approach, as it is much easier to maintain a client's state. There is no page navigation and any state changes in a widget can easily be permanently stored using native client-side data persistence APIs.

7.2.5 Maintainability

Unlike a website hosted on a central server, a widget is hosted on a client and so maintenance updates are required to update widgets to new versions. This introduces new challenges for ensuring that all clients are running the latest stable and secure versions. For a widget to be digitally signed and distributed through an official widget gallery or application store, it needs to be manually inspected, tested and validated by a third party. This process is not well suited to regular updates and so, unlike browser-based RIAs, widgets are restricted to fewer development iterations and maintenance updates.

7.2.6 Testing

Widget testing practices are in an immature state and current ad hoc development approaches do not promote the development and execution of repeatable tests. Widget testing normally requires manual simulation of scenarios or the development of custom scripts to test behaviour. There are no test suites currently available to test widget behaviour, although testing tools are being developed to validate that widgets and widget engines conform to W3C specifications [67].

7.3 Security

Web applications are more susceptible to attacks than traditional hosted applications [2]. Web systems are complex and are often developed using ad hoc processes and techniques. The majority of Web designers and developers have limited knowledge of security issues and techniques required to prevent attacks. As a result, Web-based security breaches are common [86]. Widget-based approaches oppose the very restricted sandbox environment used by Web browsers to limit potential attacks [69]. Although access to the underlying platform allows for the development of richer interactive client-side applications, exposing functionality on a client platform introduces a number of security concerns. A number of vulnerabilities exist as a widget may be configured to access the local file system, execute tasks in a command line or have the ability to make HTTP requests to multiple domains.

The standard JavaScript `eval()` method presents a significant weakness in the JavaScript language [86, 87]. When passing a string of JavaScript code to the

`eval()` method, the string is validated and executed. This method is commonly used in Ajax implementations to convert JSON encoded strings into accessible data objects. The popularity of using this technique is evident in a study conducted by Yue et al., where it was found that 44.4% of 6805 websites in a range of domains use the `eval()` method to dynamically generate and execute JavaScript code [86]. The vulnerability of this is that by passing JavaScript code to `eval()`, malicious behaviour may easily be injected into a widget at runtime. Alternative approaches exist to safely use `eval()` to parse JSON strings and prevent this kind of exploitation, however, the majority of Web developers and users are unaware of how to protect against this vulnerability [86]. To demonstrate this vulnerability the example in listing 2 and 3 is used.

Listing 2 Sample client code to perform HTTP GET and load JSON result

```
//Prepare and send HTTP request
url = http://frame-rate.appspot.com/rpc?action=
GetGenreFilms&id=testu1&genre=comedy&rated=false
request.open( "GET", url, false );
request.send();

//Check response and return de-serialised object
if (request.status == 200) {
      //Load JSON (any JavaScript in responseText is executed)
      result = eval(request.responseText);
}
```

Listing 3 Malicious shell command returned in HTTP response

```
HTTP GET
http://frame-rate.appspot.com/rpc?action=
GetGenreFilms&id=testu1&genre=comedy&rated=false
HTTP/1.1 200
runCommand("rm -rd /home/")
```

In listing 2, a local JavaScript method invokes a remote Web method and expects a JSON encoded string to be returned. If the string returned by the Web method is changed on the server-side application or through a third party interception, as

shown in listing 3, malicious code can easily be injected and run on the client when `eval()` is called. If a widget has sufficient privileges, this vulnerability may be exploited to take over a client's machine and run malicious commands.

Konfabulator's security model requires that widgets define access rights such as access to the file system, command line or particular domains. Digital signatures are used to validate the integrity of a widget distributed through an official gallery or store, but due to the dynamic nature of JavaScript, digital signatures are not sufficient to ensure that malicious code is not injected and dynamically executed by a widget at runtime. As widgets use remote Web services extensively, the safety of a widget cannot be guaranteed by the packaged scripting behaviour alone.

7.4 Portability

7.4.1 Packaging

A widget package serves as a container for all the resources and metadata of a widget. Packaging allows an entire widget resource to be easily hosted and passed between different platforms. Although incompatible conventions are used to implement packaging strategies for many implementations, packaging is considered a strength of widget-based approaches. Packaging a widget within a single unit facilitates the transport and definition of a RIA. As packages are hosted on a client, only dynamic content and behaviour is required to be transported during runtime. This is particularly attractive for mobile platforms where continuous connectivity may be restrictive. Technologies used to package widgets are open and relatively simple, and so have been identified by the W3C as a key concern that will benefit from standardisation, without restricting innovation. Packaging recommendations will likely assist in reducing many of the incompatibilities between new widget implementations as they are at a more mature state and already being adopted.

7.4.2 Cross-Platform Portability

A large portion of the code developed for the FrameRate widget depends on Konfabulator's conventions and native API. The FrameRate widget is loosely coupled from the Web service and media player, but very tightly coupled to the engine conventions. Very little code is portable to other widget engines even if they also use similar Web

standards and expose similar API capabilities. Porting the widget to the Google gadget engine for example, requires a complete rewrite of the widget client code, even though similar functionality is provided by the Google gadget implementation.

A significant concern raised by standardisation efforts, is the poor portability of widgets due to the numerous incompatibilities discussed in section 2.3.1. Although a number of efforts are under way to reduce incompatibilities, standards have not yet been completed or adopted by many existing engines. When compared to attempts to create cross-platform runtime environments such as Java, where Sun (now Oracle) is responsible for coordinating the development of Java virtual machines, widgets face much greater challenges due to the larger number of stakeholders exploring and building runtimes. Even Java has struggled to completely ensure cross-platform capabilities. Java Enterprise servers for example are built from common standards and specifications, but incompatible implementations continue to exist [88]. In order for widgets to be uniformly used for developing and deploying cross-platform RIAs, an open standardised software platform is required. This remains a major challenge for widget-based approaches.

7.5 Presentation

7.5.1 User Interface

Widgets provide the ability to clearly separate the XML definition (defining the UI and layout) from JavaScript code. The UI definition is loaded prior to any scripting behaviour and so, references are created to all UI elements. For most cases there is no need to traverse a DOM tree and UI elements may be directly accessed and manipulated by JavaScript code. The UI and scripting code are well separated and the layout can easily be altered without having to change any underlying JavaScript. Assuming the same scripting behaviour is supported by different platforms, this is advantageous, as it allows different UIs to be created for different platforms without having to change any of the underlying behaviour.

7.5.2 Styles

Although the Konfabulator engine supports a subset of CSS syntax, it does not support the inclusion of style hierarchies in a separate file to the layout and scripting.

This results in repeated styles for similar UI elements. Separation of CSS styles, as done for Web pages, will assist to further improve the decoupling of markup, styles and behaviour. This is a minor weakness that may easily be resolved.

7.6 Outcome

7.6.1 Summary of Findings

FrameRate demonstrates how widgets provide a simplified approach to develop RIAs capable of interacting with the platform and remote applications in ways that are often restricted by Web browsers. The framework developed has assisted in identifying the components required to create a RIA using a widget-based approach. The widget engine is a key component of this framework as it provides an extensive runtime environment that is able to abstract a widget from the underlying platform and any remote application services.

The main advantage identified for widget-based approaches, is the ease of development, especially for the large pool of developers used to working with Web technologies. The ease of development is assisted by the widget engine's native library that simplifies the UI, data persistence and interaction with any remote or local service. Packaging a widget resource also provides an attractive means of distributing RIAs between different platforms.

Widgets have more privileges than Web browser-based applications, but also introduce a number of security vulnerabilities. The security of widgets has been identified as a significant concern and weakness of widget-based approaches. Digital signatures are used to certify applications distributed through official widget galleries, but due to the dynamic nature of JavaScript, digital signatures are not sufficient to ensure that widgets do not behave in a malicious way at runtime. Unless security models are improved, it is likely that widget-based approaches will be limited to simple RIAs with restricted behaviour.

Many of the promises of standards-based widget development have yet to surface, and fragmentation between implementations continues to remain a significant limitation of widget-based approaches. Although the majority of widget engines make use of common Web standards, there are still many incompatibilities between different implementations. The abstraction offered by APIs is a major strength of widgets, but

APIs have been identified as the area of greatest incompatibility between different implementations. APIs are generally more difficult to standardise when compared to general concerns such as packaging.

7.6.2 The Future of Widgets

The substantial hype and development activity around the use of widget-technologies suggests that widgets will continue to be used in future to develop RIAs for different platforms. Widgets have the backing of large influential organisations across multiple industries and will continue to compete with other approaches to develop and deploy RIAs, such as those used for iPhone or Android-based applications [28, 65].

Widget standardisation efforts are ambitious in their attempts to reduce the incompatibilities of a technology that is still being actively developed and explored. Many of the limitations identified for widgets will improve through the continued collaboration between major stakeholders driving widget standardisation. The success of widgets is largely dependent on the creation of open standardised frameworks for developing and distributing widget applications. Although widget standards are still in early drafts, they will significantly influence how future widgets are developed, distributed and used in more secure and efficient ways. Unless the standardisation activities influence prominent implementations, the widget landscape will continue to be made up of numerous incompatible implementations.

The key areas identified requiring substantial attention are that of security, portability and widget APIs. Further work is required to improve existing security models if widgets are to be used for more advanced, secure applications. Platform APIs are being actively standardised through initiatives like Bondi, however, efforts to address incompatibilities between general widget APIs are limited and require further attention. Concerns such as maintainability, testing and styles are relatively minor challenges as these will be resolved and improved as widget technologies mature. As demonstrated by GWT, more structured languages such as Java may be used to develop applications compiled to JavaScript. JavaScript development environments and interpreter efficiency will continue to improve, and so it is unlikely that the use of JavaScript will restrict widget development.

7.6.3 Limitations of the Study

A limitation of the study conducted is that only a single widget has been developed for a single engine. Issues such as cross-platform portability have been identified through a review of the widget landscape, but are not demonstrated by FrameRate.

Choosing a suitable RIA development and deployment strategy is often a partially subjective decision that depends on a number of factors. Many of these factors are a result of economic drivers and strategic decisions that are not explicitly considered in the study. Factors influencing the choice of RIA approach include:

- The target platform (a particular mobile phone, television, desktop or operating systems).
- The number of devices supporting a particular runtime (browser, widget engine or other runtime such as Java).
- Functional capabilities of a particular RIA technology (security, user interface).
- Performance requirements (speed, robustness or maintenance).
- Developer skills, available tools and technical support.

Alternative approaches to developing RIAs have both advantages and limitations, but based on the scope of the work done, it cannot be concluded whether widget-based approaches are technically superior to other RIA approaches. Further work is required to compare the strengths and weaknesses of widgets to popular alternatives that include [4, 11]:

- Adobe Flash, Integrated Runtime (AIR) or Flex.
- Android, iPhone and Symbian RIAs.
- Google Web Toolkit (GWT) Ajax framework.
- JavaFX.
- Microsoft ASP.Net Ajax framework.
- Microsoft Silverlight.
- Standard Web Browser Ajax (Without using a framework).

Chapter 8

Conclusion

The ability to permanently connect different consumer electronic devices to the Internet has influenced how people interact with Web-based content, applications and services. The Web continues to evolve to support new requirements for developing and distributing new software applications on different platforms. A number of alternative approaches exist for developing what is broadly described as a RIA: an Internet-based application offering similar features and functionality to that of one hosted on a client platform. A Web browser's primary role is to allow users to navigate between different resources on the Web, and remains the principal application allowing clients on different platforms to use RIAs. The alternative approaches to using the browser as a RIA runtime, is to develop Internet enabled applications for a platform's native operating system or to use another runtime environment such as a widget engine.

Widgets have gained considerable attention and are being used to develop and distribute RIAs on multiple platforms that include mobile phones and televisions. Widgets are essentially a client-side application authored using Web standards. Widget engines provide the runtime environment and abstracted interfaces for widgets to interact with the platform in ways that are restricted by Web browsers.

A consequence of the rapid rise of widget-based technologies is that there are numerous incompatibilities between different widget engines on different platforms. Incompatibilities between widgets include issues related to development, packaging, configuration, metadata, internationalisation, maintenance and security. A number of standardisation initiatives are attempting to reduce these incompatibilities. The most notable effort being carried out to standardise widgets is that of the W3C's Web Applications working group. The group is looking to standardise general

characteristics of widgets, while platform specific concerns are being addressed by other efforts such as Bondi and JIL. Standardisation efforts are still in early phases and so a standard framework for developing widget applications has not been established. Standardisation efforts hope to create an open standard for developing and deploying widgets on multiple platforms.

Widgets present an opportunity to formalise the approaches used to develop and deploy RIAs for different platforms. The potential adoption of widgets, raises the question of whether widgets successfully meet the requirements for developing and deploying RIAs? The core requirements of widgets are identified by behaviour, ease of development, security, portability and presentation concerns.

A conceptual widget framework has been developed to describe the common functional units and architecture of widgets, independent of a particular platform or implementation. The framework developed has assisted in identifying features to meet the core requirements identified. The framework has also assisted in identifying the architecture and key subsystems of a widget solution.

A film rating and recommendation service (FrameRate) has been developed based on the conceptual widget framework. The FrameRate service demonstrates the role of widgets and consists of a content provision service, a film rating Web service and a service delivery platform hosting a widget engine and application. The service is used as a reference implementation to evaluate the suitability of widget-based approaches.

Widgets successfully meet certain requirements of RIAs, however, a number of weaknesses have been identified. The main strength of widget-based approaches are that they:

- Allow RIAs to access advanced behaviour on a platform or remote application.
- Help to reduce the complexities of developing RIAs by:
  - Allowing applications to be developed at a higher level of abstraction.
  - Allowing applications to interact with different services without having any specialised knowledge of their complexities.
  - Provide extensive libraries to commonly used functionality, such as event handlers, timers, string parsing and data persistence.
- Separate behaviour and presentation concerns.

A number of weaknesses of widgets have also been identified. These are summarised here:

- Due to the dynamic nature of JavaScript and the common usage of language features such as the `eval()` method, widgets present various vulnerabilities where malicious code may be injected into a widget during runtime and executed outside of a security sandbox.
- Due to the flexibility of the JavaScript language, JavaScript development tools are unable to detect many potential errors prior to runtime. While this may not be a concern for simpler applications, the lack of structured approaches is a concern for developing larger applications.
- There are no established testing practices and tools for widget development in JavaScript.
- As widgets are hosted on a client, maintenance updates are considerably more challenging than for centrally hosted Web applications.
- Numerous incompatibilities exist between different engines and API conventions, thus limiting portability. As widget scripting code is tightly coupled to engine API conventions, porting widgets to other engines usually requires a complete rewrite.
- Standardisation efforts to address API incompatibilities are still limited.

JavaScript will continue to remain an important language for building Web applications in future at higher levels of abstraction. Although the flexibility offered by JavaScript can be a disadvantage for building larger applications, the popularity and strategic importance of the JavaScript will continue to influence the improvement of development environments and interpreter efficiency.

The role of widgets as an application development and distribution model is largely based on a philosophy, that as widgets are developed using Web standards, they are interoperable and may be written once and run everywhere. As demonstrated by previous efforts to achieve cross-platform portability, such as Java, achieving true portability between different operating systems and platforms is extremely difficult. The incompatibilities across the widget landscape limit the portability of widgets between platforms and developers are required to choose a particular platform and engine to develop a widget for.

Developing RIAs using Web standards may suit existing Web developers, however, no overriding benefits have been identified for widget-based approaches. Widgets are suitable for developing simple applications with limited behaviour. If widgets are used for more advanced applications; security, portability, maintenance and JavaScript become limiting factors requiring further attention.

Alternative approaches to developing RIAs have both advantages and limitations, but based on the scope of the work done, it cannot be concluded whether widget-based approaches are technically superior to other RIA approaches. A complete comparative study of other RIA frameworks is required to determine the suitability or limitation of widgets compared to other RIA approaches

Following the findings of the research conducted, it is still unclear whether widgets will be adopted as a mainstream approach for developing RIAs in future. The future of widget technologies and their role as a packaged client-side Web application, largely depends on the outcomes of various standardisation efforts being carried out. Widgets may provide a simplified approach to developing RIAs, but they have not yet reached a level of maturity where this can be done across multiple platforms in a uniform and secure way. Unless widgets can address security and portability concerns, it is likely that they will continue to remain a means of developing simple applications with limited access to underlying platform capabilities.

References

[1] G. Kappel, B. Proll, S. Reich, and W. Retschitzegger, *Web Engineering: The Discipline of Systematic Development of Web Applications*, 1st ed. John Wiley & Sons, 2006.

[2] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, *Web Engineering: Modelling and Implementing Web Applications*. Springer, 2008.

[3] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz, "Web browser as an application platform," in *SEAA '08. 34th Euromicro Conference on Software Engineering and Advanced Applications*, Sep 2008, pp. 293 – 302.

[4] G. Lawton, "New ways to build rich Internet applications," *IEEE Computer*, vol. 41, no. 8, pp. 10–13, Aug 2008.

[5] T. Berners-Lee, *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, 1st ed. HarperOne, 1999.

[6] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The web of things: Interconnecting devices with high usability and performance," in *Embedded Software and Systems. ICESS '09. International Conference on*, May 2009, pp. 323–330.

[7] J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford, "Security models for web-based applications," *Communications of the ACM*, vol. 44, no. 2, pp. 38–44, 2001.

[8] Adobe Systems, "Adobe flash player," http://www.adobe.com/products/flashplayer/, Nov 2009.

[9] D. B. Delgado, "Inspiring teamwork & communication with a content management system," in *Proceedings of the 35th annual ACM SIGUCCS conference on User services*, 2007, pp. 55–59.

[10] S. Weerawarana, F. Cubera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*, 1st ed. Prentice Hall, 2005.

[11] L. D. Paulson, "Building rich Web applications with Ajax," *IEEE Computer*, vol. 38, no. 10, pp. 14–17, Oct 2005.

[12] I. Hickson and D. Hyatt, "HTML5 A vocabulary and associated APIs for HTML and XHTML," W3C Working Draft, http://www.w3.org/TR/html5/, Aug 2009.

[13] P. Mendes, M. Cáceres, and B. Dwolatzky, "A review of the widget landscape and incompatibilities between widget engines," in *IEEE Africon*, Nairobi, Kenya, Sep 2009.

[14] D. Odell, *Pro JavaScript RIA Techniques*, 1st ed. Apress, 2009.

[15] Microsoft, "ASP.net Ajax," http://www.asp.net/ajax/, Nov 2009.

[16] Google, "Google web toolkit," http://code.google.com/webtoolkit/, Nov 2009.

[17] Sun Microsystems, "JavaFX," http://javafx.com/, Nov 2009.

[18] Adobe Systems, "Adobe AIR," http://www.adobe.com/products/air/, Nov 2009.

[19] A. Jaokar and T. Fish, *Mobile Web 2.0 : The Innovator's Guide to Developing and Marketing Next Generation Wireless Mobile Applications*, 1st ed. Futuretext London, 2006.

[20] Yahoo! Widgets, "Konfabulator 4.5 reference manual," http://manual.widgets.yahoo.com/, Nov 2007.

[21] G. Lawton, "These are not your father's widgets," *IEEE Computer*, pp. 10–13, Jul 2007.

[22] M. Cáceres and M. Priestley, "Widgets 1.0: Requirements working draft," http://www.w3.org/TR/widgets-reqs/, Apr 2009.

[23] S. Nylander, M. Bylund, and A. Waern, "Ubiquitous service access through adapted user interfaces on multiple devices," *Personal and Ubiquitous Computing*, vol. 9, no. 3, pp. 123–133, May 2005.

[24] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transaction on Computer Human Interaction*, vol. 7, no. 1, pp. 3–28, 2000.

[25] M. Cáceres, "Widgets 1.0: The widget landscape (Q1 2008) working draft," http://www.w3.org/TR/widgets-land, Apr 2008.

[26] Google, "iGoogle developer home," http://code.google.com/intl/en/apis/igoogle/, Nov 2009.

[27] C. Kaar, "An introduction to widgets with a particular emphasis on mobile widgets," http://symbianresources.com/cgi-bin/schlabo/dl.pl?WidgetsTR, Oct 2008.

[28] F. Reynolds, "Web 2.0–in your hand," *IEEE Pervasive Computing*, vol. 8, no. 1, pp. 86–88, Jan–Mar 2009.

[29] H. Hanrahan, "Modelling convergence: Technology layering for horizontal regulation," in *Proceedings of Southern African Telecommunications and Applications Conference (SATNAC)*, Sep 2004.

[30] Symbian Foundation, "Symbian developer network," http://developer.symbian.org, Nov 2009.

[31] Apple, "iPhone dev center," http://developer.apple.com/iphone, Nov 2009.

[32] Opera Software, "Opera mobile browser," http://www.opera.com/mobile, Nov 2009.

[33] J. Gosling, "Java intermediate bytecodes," in *Papers from the 1995 ACM SIGPLAN workshop on Intermediate representations*. New York, USA: ACM, 1995, pp. 111–118.

[34] M. Rosenblum, "The reincarnation of virtual machines," *Queue*, vol. 2, no. 5, pp. 34–40, Aug 2004.

[35] Opera Software, "Opera widgets," http://www.opera.com/business/solutions/widgets/technology/, Nov 2009.

[36] Android, "Developers page," http://developer.android.com, Nov 2009.

[37] Apple, "Apple's revolutionary app store downloads top one billion in just nine months," http://www.apple.com/pr/library/2009/04/24appstore.html, Apr 2009.

[38] Gartner, "Market trends: Mobile devices, worldwide 2009," *Gartner Research Report*, vol. 166495, Mar 2009.

[39] N. Jones, "Nokia widgets will encourage S60 mobile services," *Gartner Research Report*, vol. G00148087, Apr 2007.

[40] Microsoft, "Developing widgets for windows mobile 6.5," http://msdn.microsoft.com/en-us/library/dd721906.aspx, Jun 2009.

[41] BlackBerry, "BlackBerry Widget SDK 1.0 beta 1," http://na.blackberry.com/eng/developers/devbetasoftware/widgetsdk.jsp, Nov 2009.

[42] V. Group, "Betavine open mobile application community," http://www.betavine.net/bvportal/resources/widgets, Nov 2009.

[43] J. F. Jensen, "Interactive television - a brief media history," in *EUROITV '08: Proceedings of the 6th European conference on Changing Television Environments, Salzburg, Austria*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–10.

[44] T. S. Perry, "The trials and travails of interactive TV," *IEEE Spectrum*, pp. 22–28, Apr 1996.

[45] S. Karlin, "Engineering entertainment," *IEEE Spectrum*, vol. 39, no. 3, pp. 69–70, Mar 2002.

[46] Intel Consumer Electronics, "Widget channel software framework technology brief: Widget channel: Personalize, enjoy and share your favorite Internet experiences on TV," http://www.intelconsumerelectronics.com/Download/320270-003US.pdf, Aug 2008.

[47] Intel Software Network, "Widget development kit (WDK) pre-release program," http://software.intel.com/en-us/articles/widget-development-kit-wdk-pre-release-program, Sep 2009.

[48] G. Cronin, "Marketability and social implications of interactive TV and the information superhighway," *IEEE Transactions on Professional Communication*, vol. 38, no. 1, pp. 24–32, March 1995.

[49] C. M. Christensen, *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*, 1st ed. Harvard Business School Press, 1997.

[50] M. L. Tushman and P. Anderson, "Technological discontinuities and dominant designs: a cyclical model of technological change," *Administrative Science Quarterly*, vol. 5, no. 4, pp. 604–634, Dec 1990.

[51] M. Cáceres, "Widget packaging and configuration candidate recommendation," http://www.w3.org/TR/widgets/, Dec 2009.

[52] Opera Software, "Opera widgets security model," http://dev.opera.com/articles/view/opera-widgets-security-model/, Feb 2010.

[53] W3C, "About the World Wide Web Consortium," http://www.w3.org/Consortium/Overview, Apr 2008.

[54] W3C Web Applications Working Group, "Working group charter," http://www.w3.org/2008/webapps/charter/, Nov 2009.

[55] I. Jacobs, "World wide web consortium process document," http://www.w3.org/2005/10/Process-20051014/, Oct 2005.

[56] R. Berjon, "Widgets 1.0: Widget URIs working draft," http://dev.w3.org/2006/waf/widgets-uri/, Oct 2009.

[57] F. Hirsch, M. Cáceres, and M. Priestley, "Widgets 1.0: Digital signatures candidate recommendation," http://www.w3.org/TR/widgets-digsig/, Jun 2009.

[58] M. Cáceres, R. Berjon, and A. Bersvendsen, "The widget interface candidate recommendation," http://www.w3.org/TR/widgets-apis/, Dec 2009.

[59] R. Berjon, "Widgets 1.0: Access requests policy (WARP) working draft," http://www.w3.org/TR/widgets-access/, Dec 2009.

[60] M. Cáceres, "Widgets 1.0: Updates working draft," http://www.w3.org/TR/widgets-updates/, Oct 2008.

[61] M. Hanclik, R. Berjon, A. Bersvendsen, and M. Cáceres, "Widgets 1.0: View modes media features working draft," http://dev.w3.org/2006/waf/widgets-vmmf/, Oct 2009.

[62] Open Mobile Terminal Platform, "Bondi and open source industry collaboration for widget and web technologies," http://bondi.omtp.org, Sep 2009.

[63] Joint Innovation Lab, "JIL developer website," http://www.jil.org, Nov 2009.

[64] Open AJAX Alliance, "Gadgets task force," http://www.openajax.org/member/wiki/Gadgets, Nov 2009.

[65] Joint Innovation Lab, "Leading handset manufacturers to support the Joint Innovation Lab (JIL) initiative," http://www.prnewswire.co.uk/newsindex.shtml?/cgi/news/release?id=269395, Oct 2009.

[66] S. Santos, D. Silva, and M. Cáceres, "Conformance matrix for widgets packaging and configuration: Work in progress," http://samaxes.svn.beanstalkapp.com/widgets_compatibility_matrix/trunk/index.html, Nov 2009.

[67] W3C Web Applications Working Group, "Widget testing," http://www.w3.org/2008/webapps/wiki/WidgetTesting, Nov 2009.

[68] M. Bishop, *Computer Security: Art and Science*, 1st ed. Addison-Wesley Professional, 2003.

[69] R. Cox, J. Hansen, S. Gribble, and H. Levy, "A safety-oriented platform for Web applications," in *IEEE Symposium on Security and Privacy*, May 2006.

[70] WebKit, "WebKit browser engine," http://webkit.org/, Nov 2009.

[71] C. Peng, A. Lugmayr, and P. Vuorimaa, "A digital television navigator," *Multimedia Tools and Applications Journal*, vol. 1, no. 17, pp. 121–141, May 2002.

[72] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, Aug 2009.

[73] A. Cockburn, *Writing Effective Use Cases (The Agile Software Development Series)*. Addison-Wesley Professional, 2000.

[74] School of Electrical and Information Engineering, "Centre for Telecommunications, Access and Services, University of the Witwatersrand," http://cetas.ac.za, Nov 2009.

[75] J. Lee, J. Kim, S. Kim, C. Lim, and J. Jung, "Enhanced distributed streaming system based on RTP/RTSP in resurgent ability," in *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, 2005, pp. 568–572.

[76] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39–59, 1984.

[77] JSON, "Introducing JSON," http://www.json.org/, Sep 2009.

[78] Google, "AppEngine," http://appengine.google.com/, Nov 2009.

[79] IMDB, "The Internet movie database," http://www.imdb.com/, Sep 2009.

[80] VideoLAN, "Building pages for the http interface," http://wiki.videolan.org/Documentation:Play_HowTo/Building_Pages_for_the_HTTP_Interface, Sep 2009.

[81] The Dojo Foundation, "Dojo Javascript toolkit," http://dojotoolkit.org/, Nov 2009.

[82] jQuery, "jQuery Javascript library," http://jquery.com/, Nov 2009.

[83] D. Flanagan, *JavaScript: The Definitive Guide, 4th Edition*, 4th ed. O'Reilly, Nov 2001.

[84] M. Chang, E. Smith, R. Reitmaier, M. Bebenita, A. Gal, C. Wimmer, B. Eich, and M. Franz, "Tracing for web 3.0: trace compilation for the next generation web applications," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Washington, USA*, 2009, pp. 71–80.

[85] R. Ward and M. Smith, "Javascript as a first programming language for multimedia students," *SIGCSE Bull.*, vol. 30, no. 3, pp. 249–253, 1998.

[86] C. Yue and H. Wang, "Characterizing insecure javascript practices on the web," in *WWW '09: Proceedings of the 18th international conference on World Wide Web, Madrid, Spain*, 2009, pp. 961–970.

[87] D. Yu, A. Chander, N. Islam, and I. Serikov, "Javascript instrumentation for browser security," in *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Nice, France*, 2007, pp. 237–249.

[88] G. Fox, "Java and grande applications," *Computing in Science and Engineering*, vol. 5, pp. 60–69, 2003.

Appendix A

Requirements

FrameRate Requirements Specification - Revision 1.0

1. Introduction

This document serves to describe the requirements specifications for developing a film rating system for television. The requirements serve to define the scope of the solution as well as future desirable features. The core requirements are defined for the first release cycle and additional requirements are provided to identify desirable features to be considered in the design and possibly included in future releases.

2. Background Information

Traditional television broadcast services have started to support a greater level of interaction for a user. The Internet provides a channel for televisions and set-top boxes to deliver richer experiences by combining traditional broadcast services with those available on the Web. Interactive television, IP TV and streaming services have undergone various cycles of innovation to deliver richer interactive experiences to users, however a standard way of achieving this interactivity has not yet emerged. A fairly recent advancement in the interactive television is to use widget applications to create these experiences.

The role of the proposed film rating system is to demonstrate and evaluate how a widget application can be used to achieve an interactive television experience. The domains of interaction are the user, television and film.

3. Terms and Definitions

Terms and definitions used throughout the requirements specification document are outlined in Table 1. Key terms and definitions used in the requirements are capitalised to clearly define components and reduce any ambiguities.

Table 1 - Terms and Definitions

Term	Definition
User	A person who watches television
Television	A Television is capable of playing content and interacting with a User. This capability may be hosted on a television or a connected set-top box and so no distinction is made between the two
Set-top Box	See Television
FrameRate	Television Rating System described in this requirements document

Term	Definition
Genre	The Genre of a film categorising its nature: Action, Comedy, Family…
Rating	A User rating from 0 to 5 where 0 is poor and 5 is excellent
Group	A group consists of 2 or more users registered on a system
Playback	The reproduction of a Film resource on a Television. Playback includes when a film is paused or playing.
Profile	A dynamic User profile which holds information about a users movie rating activities

4. Scope and Purpose

The document describes *what* functionality FrameRate is to perform, but makes no effort to describe *how* this functionality is to be implemented.

4.1.1 Responsibilities

The Requirements Spec is to be reviewed and maintained by all members of the development team. It serves as a reference throughout the project lifecycle and provides the benchmark of the functionality to be designed, developed and tested.

4.1.2 Requirements Descriptions

The requirements descriptions are worded such that each requirement is distinct, concise and measurable [1].

If a requirement no longer applies and it is removed from the product, it should still remain in the document with a strikethrough.

For example:

~~The user shall be able to enter a number between 1 and 5.~~

Each requirement shall use one on the following terms: *shall*, *will* or *may* to describe the desirability of the function. This helps to indicate future requirements that should be accounted for.

- Shall means that the functionality is mandatory and must be in the release.
- Will means that the functionality is highly desirable but not mandatory.
- May means that the functionality is desirable but should be considered as an optional extra.

4.1.3 Requirement Numbers

Each requirement must be uniquely numbered in the following format:

XX NNN

Where:

- XX: Incrementing integer [0:99] for each requirement category
- NNN: Incrementing integer [0:999] for each requirement within a category

Requirement numbers are consistent for each release cycle and should be used to identify bugs and feature requests. Intervals between requirement numbers are left so that future requirements can be introduced and grouped together with similar existing requirements.

5. Assumptions

Only films are being considered as content and the widget can operate in single window mode.

6. Requirements

6.1 Users

The rating system keeps track of film ratings for each individual per film. Various levels of users may interact with the television rating system:

- Individual User
- A Group of Users
- All Users registered on a particular system

Table 2 – User Requirements

01 001	FrameRate will allow a User to create a new profile
01 002	FrameRate will allow a User to delete a profile
01 003	FrameRate will allow a User to update a profile
01 004	FrameRate shall allow a Group of Users to be defined for a television
01 005	FrameRate will support a default Group for all Users

6.2 Rating

Table 3 – Rating Requirements

02 001	FrameRate shall allow a User, Group or All users to Rate a Film during playback
02 002	FrameRate shall allow a User, Group or All users to Re-Rate a Film during playback

6.3 Browsing

Browsing provides a means of suggesting films to a user based on their movie ratings. The browsing functionality serves to help users find films that they are likely to rate highly based on the ratings of other users.

Table 4 – Browsing Requirements

03 001	FrameRate shall allow users to browse the top 10 films, which they have not yet rated in a particular Genre.
03 002	FrameRate shall allow users to browse the top 10 films, which they have already rated in a particular Genre.
03 003	FrameRate will allow users to browse further films
03 004	FrameRate will allow users to browse the top films similar to the one playing
03 005	FrameRate will allow a users to only browse films accessible by them
03 006	FrameRate will allow users to browse films rated by other users with similar ratings

6.4 Playback

04 001	FrameRate shall allow users to select and play a film on the host system from a browser pane

6.5 Security

Security concerns serve to ensure that when a user creates deletes of updates a profile, it is authorised by the owner of that profile. These concerns are not vital to demonstrate and so are left as future requirements.

Table 5 – Security

05 001	FrameRate will provide a means of authenticating users prior to altering their profiles or ratings

6.6 Interfaces

Table 6 – Interfaces

06 001	FrameRate shall expose an interface to rate a current film being played
06 002	FrameRate shall expose an interface to browse films during playback
06 003	FrameRate shall be able to hide any interface exposed to a user during playback
06 004	Interfaces exposed shall support restricted user input (remote and no keyboard)

6.7 Logging and Error Handling

Table 7 – Logging and Error Handling

07 001	FrameRate shall notify a user when it is not able to access user profiles or content services
07 002	FrameRate may log significant events such as errors and the creation of new profiles

6.8 Performance

Requirement describing performance expectations

Table 8 – Performance Requirements

08 001	FrameRate shall not interfere with the playback or network performance of the host platform

6.9 Content

Table 9 – Content Requirements

09 001	FrameRate may support additional content to film, such as music and television series
09 002	FrameRate may link up with an existing movie library

7. Recommendations

This document should be continually reviewed in each release cycle and updated to identify all the features that should exist and be tested in each release cycle.

8. Conclusion

The requirements presented identify the core features required, but also highlight desirable features that may be introduced in future versions.

References

[1] Electric Power Research Institute “Software Requirements Document & Review” http://mydocs.epri.com/docs/SDRWeb/processguide/srd.html, last accessed 19 March 2008

Appendix B

Functional Specifications

FrameRate Functional Specifications - Revision 1.0

1. Introduction

This document serves to describe the Functional Specifications for developing a Film Rating System: FrameRate. The functional specification provides further details on items described in the Requirements Specification Documentation [1]. This document describes the essential system and subsystem behaviours needed to meet the requirements of FrameRate.

1.1 Scope and Purpose

The focus of the document is to outline what FrameRate needs to do rather than how it will done. The Specifications do not represent the final system developed, but serve as a design tool to identify system behaviour requirements.

2. Terms and Definitions

Terms and definitions used throughout the requirements specification document are outlined in Table 1. Key terms and definitions used in the Use Cases are capitalised to clearly identify components and reduce any ambiguities.

Table 1 - Terms and Definitions

Term	Definition
Actor	An external entity that interacts with the system. A Primary actor uses the system directly, performing one or some of the main tasks A Secondary actor supervises or maintains the system.
FrameRate	Television Rating System described in this requirements document
Genre	The Genre of a film categorising its nature: Action, Comedy, Family…
Group	A group consists of 2 or more users registered on a system
Playback	The reproduction of a Film resource on a Television. Playback includes when a film is paused or playing.
Post-Conditions	Describes the state of the system at the end of the use case execution.
Pre-Conditions	List any activities that must take place, or any conditions that must be true before the use case can be started.
Profile	A dynamic User profile which holds information about a users movie rating activities
Rating	A User rating from 0 to 5 where 0 is poor and 5 is excellent

Term	Definition
Set-top Box	See Television
Television	A Television is capable of playing content and interacting with a User. This capability may be hosted on a television or a connected set-top box and so no distinction is made between the two
Use Case	A Use Case is a description of a set of sequences of actions that a system performs to produce an observable result.
Use Case Model	A model that describes a system's functional requirements in terms of use cases. Consists of all the actors of the system and all the various use cases by which the actor interacts with the system, thereby describing the total functional behaviour of the system.
User	A person who watches television

2.1 Symbols Used

The symbols used for use case text descriptions are shown in

Table 2.

Table 2 – Use Case Symbols

Symbol	Description
	User Goal
	Summary
	Sub Function
	Computer System
	Company
	Sub Function

3. System Overview

The system being developed is a software implementation of a Film Rating System (FrameRate). The objectives of FrameRate are that it allows users to rate films that they are watching so that they are able to build a personalised history of films that they enjoy. Using this rating history, users are able to browse through recommended films for particular genres that they are likely to enjoy and have not watched.

4. Functional Specifications

The use case diagram for the system is illustrated in Figure 1.

4.1 Use Case Diagram

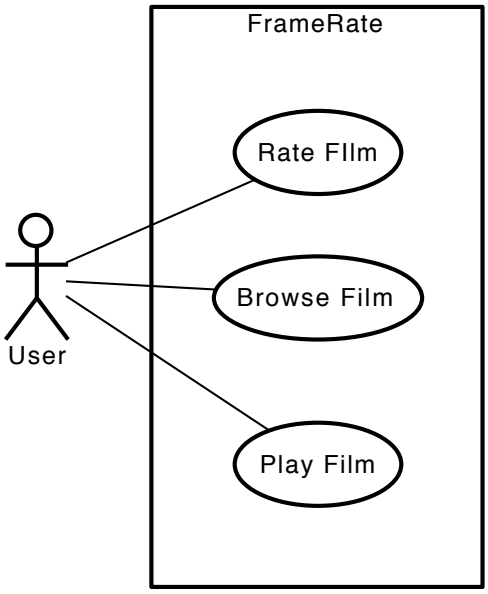
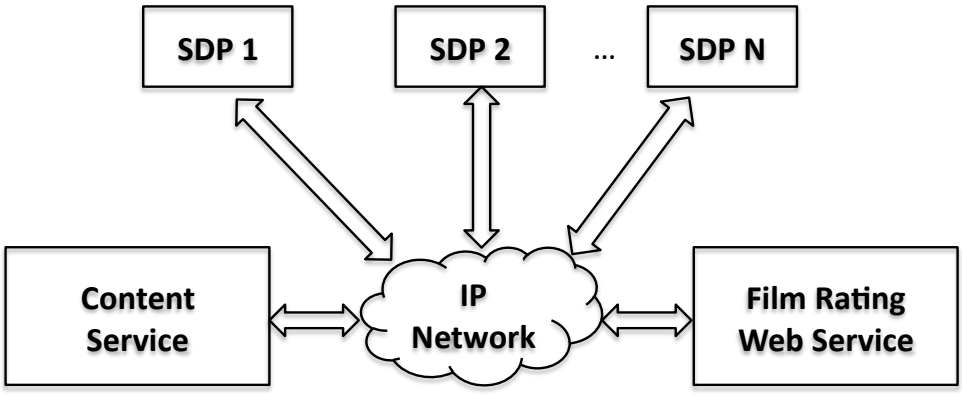


Figure 1 - Use Case Diagram for FrameRate

4.2 System

The system described is the FrameRate system. This includes all the functionality to handle User Profiles, Film Rating and Browsing Functionality. The system as illustrated in Figure 2 and consists of multiple service delivery platforms, a network, content provision service and film rating Web service.



SDP: Service Delivery Platform

Figure 2 – FrameRate System Overview

4.3 System Actors

4.3.1 User

Primary Actor. This actor represents a User who interacts with the FrameRate system to rate a film based on their preferences. As multiple users may simultaneously interact with the system, the User actor may represent a single User, a Group of Users or All Users which interact with the system.

4.4 Use Cases

The use cases developed use the template provided in [2]. A condensed version of the template is used, and further details will be added to the use cases as the system design evolves.

4.4.1 Use Case 1: Rate Film

Primary Actor: User (or Group of Users)
Scope: FrameRate SDP and Web Service
Goal in Context: A User or Group of Users rates a film to build up their rating profile
Level: User Goal
Preconditions: Users are registered on the system. Users have identified the film to be rated.
Success End Condition: User rating is captured for a particular film
Failed End Condition: The User rating is not captured by FrameRate and the film remains unrated for that User
Trigger: A User makes a request to rate a film

MAIN SUCCESS SCENARIO

1. User is watching film to be rated
2. FrameRate has determined the identity of the film being rated
3. FrameRate generates a form to capture the Rating, from 1 to 5
4. The film rating is captured and stored in a Users' profile

EXTENSIONS

1a. User is Browsing Films (Use Case 2) that have or have not already been rated
2a. The identity is selected from a list of films when Browsing Films
4a. Captured rating overrides any previous rating

SUB-VARIATIONS

1. User is watching a film that they have already rated
3. The Client may choose to omit rating: no rating is captured in their profile

4.4.2 Use Case 2: Browse Films

Primary Actor: User (or Group of Users)
Scope: FrameRate SDP and Web Service
Goal in Context: A User wished to browse through a list of recommended films in a particular genre based on their personal rating history.
Level: User Goal
Preconditions: User is registered
Success End Condition: A User is presented with a list of films matching their criteria
Failed End Condition: A film list isn't presented to the User
Trigger: A User makes a request to browse films

MAIN SUCCESS SCENARIO

1. FrameRate captures the Genre of film which a User wished to browse
2. FrameRate captures whether a User would like to browse films that have already been rated by them
3. A User requests a list of films
4. FrameRate fetches a list of films and their ratings ordered from high to low
5. FrameRate displays a subset of film titles and their ratings ordered from high to low

EXTENSIONS

1a. The genre is captured from the current film
    1a1 Default genre is selected to match current film being played
5a. A User chooses to rate a film in the list
    5a1. User selects film to be rated
    5a2. Rating is captured and submitted: Rate a Film (Use Case 1)
5b. A User is able to re-rate a film in list
5c. User requests further results
    4c1. A new list is presented to the User

SUB-VARIATIONS

3. FrameRate is unable to process the request
4a. User chooses to Plays Film on the list
4b. FrameRate is unable to capture and process film rating
4c. A user has not rated any films and has a bare profile.
5. Film list returned is invalid

4.4.3 Use Case 4: Play Film

Primary Actor: User (or Group of Users)
Scope: Content Service and FrameRate SDP
Goal in Context: A User plays a desired film
Preconditions: A User has been presented with a list of films.
Success End Condition: The desired film is played on the User's SDP
Failed End Condition: Film is not played.
Trigger: A User chooses the desired film to play

MAIN SUCCESS SCENARIO

1. User Selects desired film
2. Film plays on the users SDP

EXTENSIONS

1a. Another Film is playing
    1a1. Request confirmation from User to play selected film
1b. Desired Film is already playing
    1b1. Notify User and ignore request

SUB-VARIATIONS

1. Desired Film is not available
2. Error occurs in fetching Film

References

[1] P. Mendes *FrameRate Requirements Specification*, Version 1.0, May 2009

[2] Cockburn A.A.R., *Resources for Writing Use Cases* http://alistair.cockburn.us/index.php/Resources_for_writing_use_cases, Last Accessed 4 May 2008

Appendix C

Papers

C.1 A Review of The Widget Landscape and Incompatibilities Between Widget Engines

Presented at the Africon Conference in Nairobi Kenya. August 2009

A review of the widget landscape and incompatibilities between widget engines

Paco Mendes
School of Electrical and
Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
Email: paco.mendes@students.wits.ac.za

Marcos Cáceres
Opera Software ASA
Widget Platform Architect
Oslo, Norway
Email: marcosc@opera.com

Barry Dwolatzky
Joburg Centre for
Software Engineering
University of the Witwatersrand
Johannesburg, South Africa
Email: barry.dwolatzky@wits.ac.za

***Abstract*—A widget is a packaged interactive client-side application, commonly developed using Web standards and techniques to access data services both on the Web and host devices. The purpose of this paper is to review the widget landscape and present key characteristics and concerns of widget technologies. Various incompatibilities exist between widgets and the proprietary engines hosting them on different platforms. Incompatibilities can be classified by how different implementations handle issues related to packaging, behaviour, security and presentation. These common concerns, together with economic drivers have given rise to numerous standardisation efforts such as those being carried out by the W3C in drafting the widget family of specifications.**

I. Introduction

Although the Web browser continues to serve as the primary means of interacting with the Web, a growing need to combine services provided by a host device (such as location and cameras) with a range of Web-based services, has given rise to a new class of application known as a *widget*. Unlike a Web page, a widget is a full-fledged interactive client-side application, which provides a programmatic means of accessing data services both on the Web and host device [1]. Widgets typically have simple interfaces, yet they provide a powerful means of creating personalised user experiences by integrating the capabilities of a host device with Web applications. Services such as location awareness may be used in conjunction with services providing weather forecasts, tourism information or social networking capabilities. Widget technologies are capable of providing a means of achieving this functionality, but there is substantial fragmentation between different implementations and this has led to various standardisation efforts.

The objective of this paper is to survey key features of widget technologies and provide an overview of incompatibilities between mainstream widget engines, characteristics, concerns and standardisation efforts. The rest of this paper proceeds as follows: Section II provides a background to the role of widgets and widget engines. Section III highlights some of the incompatibilities of different widget engines, general characteristics and concerns. Lastly, Section IV provides an overview of standardisation efforts.

An alternative use of the term widget is as a *Widget Toolkit*. This is a collection of elements forming part of a Graphical User Interface (GUI) framework as described in [2] and does not fall within the context of this paper.

II. Background

The extensive adoption and use of the Internet and World-Wide-Web-based services continues to significantly influence the broader software development landscape [3]. The Web serves both as a popular and strategic environment to deploy new software systems and deliver applications for different platforms. The Web has undergone a number of evolutionary phases to support a growing need for richer networked applications, which allow for: social networking, multimedia delivery and desktop style collaborative word processors and spreadsheets. This sustained demand for multi-platform Rich Internet Applications (RIA) continues to encourage the use of new technologies to meet an ever-increasing scope of requirements [3].

Web content and applications have traditionally been served to requests from a Web browser, however, the growing strategic importance of including different aspects of the Web on multiple platforms has resulted in new ways of distributing and interacting with Web content and applications.

A. The Web Browser and Beyond

The Web browser continues to be the predominant application allowing users to access and interact with Web documents and applications. Browsers implement open Web standards and their usefulness results from users on different platforms having the ability to interact with different resources on the Web [3]. The architecture of Web browsers has however, hardly changed since Tim Berners-Lee conceptualised them nearly twenty years ago [4]. Web browsers are designed to run on stationary devices, where Web content has very restricted access to data and resources on the client. Even the Web browsers which have been migrated to mobile devices, continue to provide functionality comparable with their desktop counterparts.

The shift towards converged service offerings has seen an improvement in the processing power and capabilities of Internet enabled devices. It is not uncommon now to have a phone with one or two cameras, bluetooth, GPS, tilt-sensors

and Internet access. As demonstrated by Apple's iPhone Application Store, from April 2009 over 1 billion iPhone applications were downloaded over nine months [5]. There is significant demand from both developers and consumers to access device capabilities that integrate with Web services outside of a conventional Web browser interface. The applications being distributed through Apple's AppStore are however restricted to run on a single platform, the iPhone OS, which constitutes around 20 million devices [6].

Regardless of Apple's limited market share, Apple has demonstrated that there is significant revenue potential from providing a development platform that enables programmers to access device capabilities and Web services. As Apple's platform is proprietary and closed however, applications cannot be shared across different platforms. In order to share applications across multiple platforms and devices, an open standardised software platform is needed.

B. The Role of Widgets

Aside from economic demands, the inception of widget applications is largely driven by a requirement to provide simple interfaces to regularly access a set of personalised services without the need for a full-fledged browser. The widget approach can be considered to be an alternative rather than a new way of interacting with the Web. Widgets have gradually taken the place of traditional single-purpose applications on a user's desktop for a range of operating systems such as Apple's Dashboard or Microsoft's Gadgets. Widget applications typically provide simple interfaces to services on the host device, such as clocks, battery gauges, notepads and application controllers; or interfaces to Web-based services such as news and weather feeds, currency exchange rates, email and Web photo albums. A user is able to regularly interact with a personalised widget inventory acquired from an extensive collection of both commercial and freely available widgets.

Jaokar and Fish define a widget as "a downloadable, interactive software object that provides a single service such as a map, news feed etc." [7]. Widgets are however, not restricted to providing a single service and an application such as a map widget could also provide the weather for a user's current location. An alternative and possibly more appropriate definition of widgets is that they are "full-fledged client-side applications that are authored using Web standards and packaged for distribution" [8]. Widgets are often perceived to be simple client-side applications, but when used in service-oriented architectures it is possible to create rich applications and user experiences.

A widget is packaged as a *widget resource* for the purpose of distribution and deployment as described in [9]. Certain characteristics of the widget approach are similar to those of a Java Applet, except that widgets are authored using Web technologies and not indented to run within a Web browser [9].

The prevailing platform currently supporting widgets is the personal computer, but there has been a move towards supporting widgets across a range of consumer electronic devices [10], [11]. Many features of widgets are commercially and strategically desirable and there is much hype around the use of widgets to deliver Web applications across a range of service delivery platforms, such as the desktop, mobile phone, television or set-top box as illustrated in figure 1a.

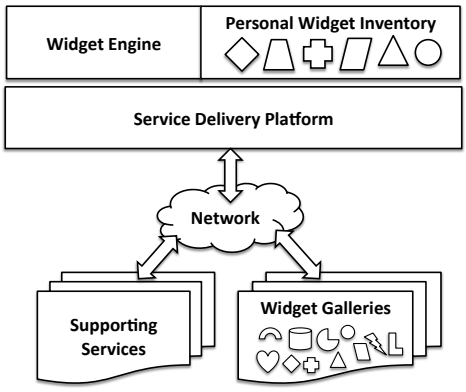
C. Widget Engines

The widget engine is the application responsible for instantiating and running a set of widgets in a user's widget inventory as illustrated in figure 1b. The widget engine provides the interfaces for a widget to interact with the underlying service delivery platform and network. Widget engines decouple a widget from the host platform and typically provide a means of accessing device specific capabilities and resources through proprietary API's (Application Programming Interfaces) and configuration files. In certain aspects, widget engines mimic the behaviour of Web browsers and many are built directly on top of Web browser frameworks to incorporate functionality such as rendering HTML mark-up and interpreting client-side scripting.

D. Differences to Web Widgets

It is necessary to differentiate a *Web widget* from the widgets considered in this paper. A Web widget is composed of fragments of HTML, CSS and ECMAScript that is dynamically or declaratively included in another Web document prior to it being served to a client [9]. Common examples are iGoogle Gadgets or Windows Live Gadgets. As widgets are authored using many of the same Web technologies as Web widgets, functional similarities exist, however there are

(a) Instantiated Widgets Running on Different Service Delivery Platforms



(b) High-level Architectural Overview of Widget Service Delivery Platforms, Galleries and Supporting Services

Fig. 1: Widget Architecture Overview

significant differences in the packaging format, security model, APIs, internationalisation and localisation. For this reason, only widgets hosted and instantiated on a client platform as illustrated in figure 1 are considered in this paper.

III. Addressing Incompatibilities

The widespread adoption of widgets and widget engines has raised a number of issues for users, developers, vendors and new entrants to the market. Widget engines on different platforms face similar challenges and provide comparable implementations, however, the rapid evolution of widgets has resulted in a number of incompatible engines. These incompatibilities are a natural consequence of any new market, where vendors compete by creating new products with new features and services to differentiate their offerings [12]. As described by Tushman and Anderson [13], new technologies undergo an era of ferment in the technological life cycle prior to the emergence of a dominant design. A dominant design for a packaged Web application is yet to emerged.

Many implementations ignore or have yet to resolve certain technical issues relating to: development, packaging, distribution, configuration, metadata, maintenance, accessibility, security internationalisation, localisation and device independence. A significant limitation of proprietary widgets is that a user cannot run a widget developed for one widget engine on another widget engine without significant modification to either the widget or the widget engine [14]. These incompatibilities have the potential of restricting widgets from becoming globally ubiquitous and so, numerous efforts are being carried out in an attempt to standardise various aspects of widgets. These efforts are discussed further in Section IV.

The growing number of platforms looking to support widgets raise various questions related to how widgets should be implemented, as well as what aspects may benefit from standardisation. Areas of incompatibility are discussed here in order to highlight common characteristics, concerns and limitations of current implementations.

A. Areas of Incompatibility

Widget engines are generally incompatible with one another as discussed in the draft widgets landscape document published by the W3C [9]. The document evaluates and compares a selection of mainstream widget engines to identify areas of fragmentation. Engines evaluated include: Yahoo! Konfabulator, Windows Vista Sidebar, Google Desktop, Opera Widgets, Apple Dashboard, Nokia Web-Runtime for the S60 mobile platform and Joost Widgets. Some of the key findings are highlighted here:

1) Development: The development approach to author widgets is generally similar across different engines. Widgets differ from traditional statically bound binary applications in that they are authored using Web technologies and techniques. With the exception of Google Desktop, all engines support HTML and CSS for the layout. All engines support common graphical resources (PNG, GIF, JPEG), scripting (Javascript) and the XMLHttpRequest object for asynchronous requests over HTTP [9].

The area of greatest incompatibility in developing widgets for different engines is the difference between the APIs allowing a widget application to handle events, errors, make use of metadata, and access host system resources and applications. Examples of implementation differences are: how a widget determines the locale, opens a URL in the host system's browser, accesses configuration details and handles events, such as when a widget switches display modes.

2) Packaging and Distribution: A key characteristic of a widget is that it is not a compiled binary file, but rather a single package, which includes any mark-up, styles, client-side scripting behaviour, and supporting multimedia resources. The main motivation for packaging a widget resource is to support use-cases such as: hosting a library of widgets for users to browse, or allowing users to pass widgets between different client platforms. A common approach used by existing engines is to package a widget resource in a Zip archive. Incompatibilities in the packaging conventions include:

- Inconsistent file extensions
- Inconsistent Internet media types
- Undefined Zip specifications
- Inconsistent packaging structure

Examples of incompatible implementation details for widget extensions and media types are provided in Table I [9].

TABLE I: Example File Extensions and Media Types

Widget Engine	Extension	Media Type
Google Desktop	.gg	app/gg
Konfabulator	.widget	application/vnd.yahoo.widget
Web-Runtime	.wgz	application/x-nokia-widgets

3) Configuration and Metadata: Widget packages typically include a structured configuration file, which holds metadata about that widget (such as the author and description) and configuration parameters (such as start-up behaviour, required resources and dimensions). All engines evaluated use XML for their configuration files [9]. Although the semantics captured are similar, there is a lack of consistent fields, namespaces and configuration parameters in the schemas.

4) Maintenence Updates: Unlike a website hosted on a central server, a widget is hosted on a client and so maintenance updates are required to upgrade distributed widget resources to new versions. There is a general lack of support for maintenance updates across different widget engines. Konfabulator implements this using a unique identifier and version number so that it can check for new versions and allow a client to download and install a new version, while maintaining the previous version's preferences [9], [15].

5) Security Models and Digital Signatures: Various incompatibilities exist as to how security policies are enforced by widget engines if they exist at all. The role of a security model is to provide policies for what actions instantiated widgets are able to perform. These policies are generally relaxed compared to those of Web browsers. Widgets are typically

able to read, write, modify and delete files; automatically upload and download files; execute local applications and perform cross-domain requests [9]. While this allows for very powerful client-side applications to be authored using Web techniques, it presents various vulnerabilities and threats to the user. Opera widgets for example follow a very tight security model adhering closer to that of a browser, while others are more relaxed [16].

Some widget engines support digital signatures to authenticate the integrity of a widgets' contents from the time that it is signed. There is inconsistency as to how digital signatures are implemented and used by different engines and so this continues to remain an area of significant fragmentation. Yahoo! for example, independently sign widgets developed for the Konfabulator engine using the Yahoo! root certificate [9].

6) Internationalisation and Localisation: Internationalisation allows a widget to operate in multiple languages without a need to significantly re-engineer the core application logic and structure. Localisation allows a widget to behave according to the location of the host client, such as setting the default home city for a weather widget. Mainstream engines are typically able to acquire the host system's locale and support a sub directory based internationalisation strategy, where content and configuration files for different languages are placed in predefined sub-directories [8]. Inconsistencies exist due to the different conventions used for the packaging structure and configuration files.

7) Device Independence: Device independence refers to how widgets can run on multiple devices. Kaar demonstrates cross platform incompatibilities by attempting to port a RSS reader widget created for Dashboard to the S60 platform [14]. It is shown that significant modification is required to port a Dashboard widget resource onto the S60 platform. Incompatibilities encountered in the investigation include:

- Incompatible configuration files
- Inconsistent use of the local file system
- Access to platform specific binaries and multimedia files

B. General Characteristics and Concerns

The areas of incompatibility highlighted in Section III-A do not describe all of the incompatibilities across the widget landscape, but provide an overview of some of the core issues faced by developers, engine vendors and users. As widget technologies continue to evolve and different engines introduce new features, it is natural that further inconsistencies will emerge. The purpose of investigating incompatibilities is to help identify common concerns and characteristics suitable for standardisation.

Perhaps a more general view of a widget is that it is an updatable structured *package* containing *presentation* resources and *behaviour* logic capable of interacting with a set of APIs on the host platform in a *secure* way. An overview of the general concerns and their relationships is illustrated in figure 2, where the key concerns identified are packaging, behaviour, security and presentation. An overview of these concerns is discussed here:

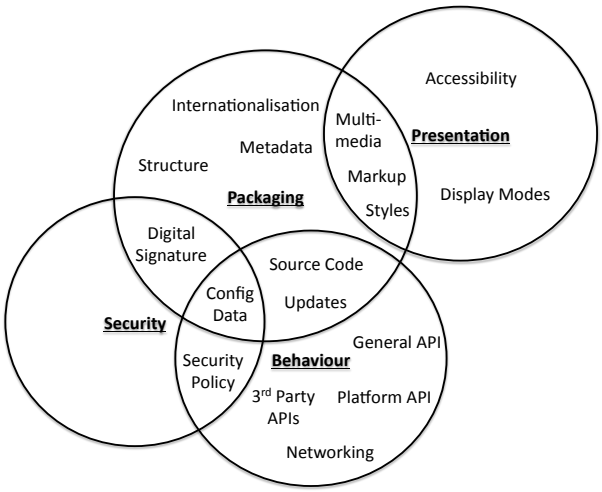


Fig. 2: General Characteristics and Concerns of Widget Technologies

1) Packaging: The conventions used to package a widget resource is a core concern for all widget engines on different platforms. With reference to figure 2, packaging encompasses issues on how source code, configuration files and the structure should be implemented and archived in a consistent way. The structure as described here, refers to the way paths to resources and configuration files are defined. Resources include configuration files, metadata, source code, mark-up, styles and multimedia [1]. Issues such as internationalisation rely on conventions specifying where resources for different languages are located. A package should also be capable of supporting a digital signature to provide a widget engine with a mechanism to certify and validate the integrity of a package's contents [1].

2) Behaviour: The behaviour of a widget resource as described here, refers to how a widget interacts with the host platform, user and any networked resources. A typical behaviour concern is how maintenance tasks are performed to update a widget package to a new version while maintaining any saved preferences. Networking is not restricted to the Internet as this behaviour can be achieved on different networks for different platforms. The source code within a widget package interacts with the host through a set of APIs. Three kinds of APIs have been identified:

- *A General API*, refers to functionality common to all widgets on all platforms such as: getting and setting user properties, updating preferences, opening a URL or closing a widget [17].
- *A Platform API*, refers to functionality for a particular platform. Functionality may include accessing common services on a mobile phone such as a camera, phone book or SMS [1].
- *A 3rd Party API*, refers to additional functionality pro-

vided to extend the General and Platform APIs.

3) Security: A widget engine should be able to determine the integrity of a package through a digital signature and enforce a security policy based on a widget's requirements [1]. Different widgets may have different security profiles based on their behaviour requirements. These may include the ability to perform cross-domain requests, access local files and interact with a hosted 3rd party application. Configuration data serves to specify the behaviour a packaged widget requires so that a widget engine is able to ensure that only authorised behaviour occurs.

4) Presentation: Presentation concerns refer to how a running widget is displayed through the use of mark-up, styles and supporting multimedia (such as video or pictures). Accessibility provides a means of allowing people with different disabilities, the ability to perceive, understand, navigate, and interact with widgets. Different display modes may exist for widgets in their different states, such as when they are unavailable, in full screen, docked or open.

IV. Standardisation Efforts

In 2005, the World Wide Web Consortium (W3C) identified widget-like technologies, which they initially called "rich Web clients", as relevant to the future use of the Web and subsequently formed the Web Applications Formats Working Group [18]. One of the main objectives of the W3C is to provide a vendor-neutral space for the development of patent-unencumbered standards, technologies, and guidelines that serve the global marketplace. In doing so, the W3C hopes to "lead the Web to its full potential" and ensure that the Web remains a royalty free medium for delivering information in a uniform way to benefit any user on any computing system [19]. The W3C is an open "pay-to-play" consortium, requiring members to pay a fee to join any standardisation effort. Despite its mantra of openness, many W3C working groups operate in secret.

A. Process

The process being followed by the W3C to standardise widgets is illustrated in figure 3. All tasks feed back on each other as the landscape is constantly changing and responding to disruptive innovations and new requirements from the Working Group's members. The standardisation process operates under a consensus model, where agreement is reached through collaboration between implementers, developers and other consortia. Collaboration is carried out using the W3C's public mailing lists and industry funded events and workshops.

The task of requirements gathering involves actively working with various stakeholders to formalise a mandatory and optional set of features that should be specified. These are formally outlined in the draft W3C's Widgets 1.0: Requirements [1]. Requirements are derived largely from a landscape analysis, which serves to identify common concerns across the widget landscape. Specifications for these requirements form part of the W3C's *Widget Family of Specifications* [20]:

- Widgets 1.0: Packaging and Configuration
- Widgets 1.0: Digital Signatures
- Widgets 1.0: API and Events
- Widgets 1.0: Updates
- Widgets 1.0: Window Modes

B. Complementary Efforts

Restrictions imposed by the W3C's Web Application Working Group Charter, limit what aspects of widgets can be standardised by the working group [21]. The W3C's standardisation efforts explicitly lack a security model, integration with Web pages as well any APIs to access device capabilities. To fill these gaps, a number of complimentary standardisation efforts are concurrently being undertaken by various consortia. They include:

- Open Mobile Terminal Platform's (OMTP) BONDI initiative [22]
- Open Ajax Alliance's Gadgets and API Task Force [23]
- Join Innovation Lab's (JIL) widget platform [24]

OMTP is a closed "pay-to-play" consortium controlled primarily by mobile operators and mobile device manufactures. In all, OMTP has nearly 40 participating companies, which include some of the most influential telecommunication industry players such as: AT&T, Vodafone, Nokia and Sony-Ericsson. OMTP's BONDI initiative is focusing on defining a security policy language for widgets, based on OASIS eXtensible Access Control Markup Language (XACML), and APIs to access device services such as the ability to send SMS, make a phone call, take a picture, access the media gallery and get the device's status [22]. In all, BONDI defines eleven APIs and relies on the W3C's Widget Family of Specifications for all other general functionality.

The Open Ajax Alliance is an open, free-to-join, consortium of companies, focusing on standardising various development aspects of the Ajax development methodology and tools. Some prominent members of the alliance include Microsoft, Adobe, IBM and the Mozilla foundation. The Open Ajax Alliance's Gadgets and API Working Groups are working on standardising Web-based widgets that are deployed on the server side (similar to Apache's Shindig gadget server technology).

Another noteworthy widget-related standardisation effort taking place is the JIL widget platform. JIL is a collaboration between Vodafone, ChinaMobile, Verizone and SoftBank [24].

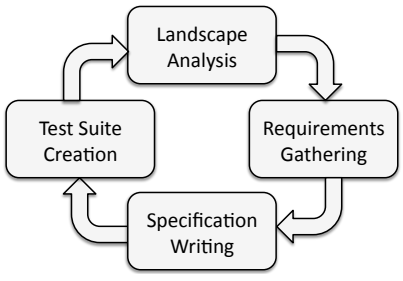


Fig. 3: Overview of the W3C widget standardisation process

JIL's standardisation activities are closed and details about what they are working on are limited to public relations documents. According to a recent public relations release, JIL is focused on "creating a single global platform for developers to encourage the creation of a wide range of innovative and useful mobile widgets" with a reach of "approximately one billion customers" [24].

C. Benefits

As noted by Jaokar and Fish, "Widgets harmonise applications development across the Web and enable a wider application distribution model" [7]. For the widget market, standardisation has some of the following potential benefits:

- Users will have to install fewer widget engines on their computers or mobile devices, thus reducing vendor lock-in and user inconvenience.
- It will make it easier for new vendors to enter the market by providing them with a complete specification from which to build a standards-based widget engine.
- Developers will have access to a larger consumer market if they build their widgets to conform to the standard.
- Widgets that, rather than serving the needs of one business, are standardised to be free, open, and unencumbered by software patents: created around principles that are aimed to benefit humanity as a whole.

V. Conclusion

There has been a significant drive by industry to integrate Web applications across a broad range of platforms through the use of widgets. Widgets provide a means of combining services on a host device with services on the Web. This move towards adopting widgets on various platforms has resulted in a rather loose definition as to what constitutes a widget, how it is developed, delivered and ultimately used. Numerous incompatibilities exist between the various widget engines available for different platforms and common concerns can be broadly categorised by packaging, behaviour, security and presentation issues. The future of widget technologies and their role in delivering packaged client-side Web applications will largely depend on the outcomes of various standardisation efforts under way. Central efforts influencing the standardisation of widgets include the W3C widget family of specifications, OMTP BONDI, the Open Ajax Alliance and JIL.

Acknowledgement

The authors would like to thank members of the Web Application working group in the W3C and volunteers who have contributed towards the various widget standardisation efforts.

References

[1] M. Cáceres and M. Priestley. (2009, apr) Widgets 1.0: Requirements. W3C working draft. [Online]. Available: http://www.w3.org/TR/widgets-reqs/

[2] F. Zammetti, *Practical JavaScript, DOM Scripting, and Ajax Projects*, 1st ed. Apress, 2007.

[3] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz, "Web browser as an application platform," in *SEAA '08. 34th Euromicro Conference Software Engineering and Advanced Applications*, September 2008, pp. 293 – 302.

[4] T. Berners-Lee, *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, 1st ed. HarperOne, 1999.

[5] Apple. (2009, apr) Apple's revolutionary app store downloads top one billion in just nine months. [Online]. Available: http://www.apple.com/pr/library/2009/04/24appstore.html

[6] Apple. (2009, apr) Apple reports second quarter results. [Online]. Available: http://www.apple.com/pr/library/2009/04/22results.html

[7] A. Jaokar and T. Fish, *Mobile Web 2.0 : The Innovator's Guide to Developing and Marketing Next Generation Wireless Mobile Applications*, 1st ed. Futuretext London, 2006.

[8] M. Cáceres. (2009, feb) Widgets 1.0: Packaging and configuration W3C working draft. [Online]. Available: http://www.w3.org/TR/widgets/

[9] ——. (2008, apr) Widgets 1.0: The widget landscape (Q1 2008) W3C working draft. [Online]. Available: http://www.w3.org/TR/widgets-land

[10] N. Jones, "Nokia widgets will encourage S60 mobile services," *Gartner Research Report*, vol. G00148087, apr 2007.

[11] Intel Consumer Electronics. (2008) Widget channel software framework technology brief: Widget channel: Personalize, enjoy and share your favorite Internet experiences on TV. 320270-003US.pdf. [Online]. Available: http://www.intelconsumerelectronics.com/Download/

[12] C. M. Christensen, *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business School Press, 1st ed. 1997.

[13] M. L. Tushman and P. Anderson, "Technological discontinuities and dominant designs: a cyclical model of technological change," *Administrative Science Quarterly*, vol. 5, no. 4, pp. 604–634, dec 1990.

[14] C. Kaar. (2008, oct) An introduction to widgets with a particular emphasis on mobile widgets. kaar07widgets.pdf. [Online]. Available: http://www.symbianresources.com/tutorials/techreports/widgets/

[15] Yahoo! Widgets. (2007, November) Konfabulator 4.5 reference manual. [Online]. Available: http://manual.widgets.yahoo.com/

[16] Opera Software. (2008, may) Opera widgets security model. [Online]. Available: http://dev.opera.com/articles/view/opera-widgets-security-model/

[17] A. Bersvendsen and M. Cáceres. (2009, apr) Widgets 1.0: APIs and events W3C working draft. [Online]. Available: http://www.w3.org/TR/widgets-apis/

[18] W3C Web Application Formats Working Group. (2005, nov) Working group charter. [Online]. Available: http://www.w3.org/2006/appformats/admin/charter.html

[19] W3C. (2008, apr) About the World Wide Web Consortium (W3C). [Online]. Available: http://www.w3.org/Consortium/Overview

[20] W3C Web Applications Working Group. (2009, apr) Working group wiki. [Online]. Available: http://www.w3.org/2008/webapps/wiki/Main_Page

[21] ——. (2009, apr) Working group charter. [Online]. Available: http://www.w3.org/2008/webapps/charter/

[22] Open Mobile Terminal Platform. (2009, apr) BONDI - and open source industry collaboration for widget and web technologies. [Online]. Available: http://bondi.omtp.org

[23] Open AJAX Alliance. (2009, apr) Gadgets task force. [Online]. Available: http://www.openajax.org/member/wiki/Gadgets

[24] Joint Innovation Lab. (2009, apr) Verizon wireless to join China Mobile, SOFTBANK and Vodafone in creating the largest global platform for mobile developers. [Online]. Available: http://www.jil.org/pr-20090402.jsp

C.2 Work in Progress Paper: A Film Rating Service Developed using a Widget-based Approach

Work in progress paper submitted to the South African Telecommunications Networks and Applications Conference (SATNAC), August 2009, Swaziland. Work presented during a poster session.

A film rating system for television developed using a widget-based approach

Paco Mendes and Barry Dwolatzky
Centre for Telecommunications Access and Services
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
Tel: +27 11 717 7204 Fax: +27 11 403 1929
Email: paco.mendes@students.wits.ac.za, barry.dwolatzky@wits.ac.za

***Abstract*—A widget is a packaged, interactive client-side application, authored using Web standards and capable of accessing local and remote data services. Widgets have become an alternative approach for developing interactive television (ITV) applications that make use of Web based services. A film rating system is being developed using a widget-based approach. A custom platform API and supporting Web service is being implemented to emulate a distributed ITV application. The goals of this research are to evaluate widget approaches to distributed software development and to model the relationship between the key sub-systems of a widget-based ITV application.**

I. Introduction

Various attempts have been made to combine traditional broadcast services with those of interactive distributed Web applications [1], [2]. There is still some uncertainty as to how entertainment services should best be delivered in future and what role the Internet and Web will serve in providing a channel of information to support interactive and personalised user experiences. One such approach is to support widget applications on a television platform.

Numerous definitions exists to describe what constitutes *a widget*. The definition adopted for this paper and current research is based on that of the World Wide Web Consortium (W3C). The W3C's widget family of specifications define a widget as a packaged interactive client-side application, developed using Web standards and techniques [3]. Unlike a Web page, a widget is a full-fledged interactive client-side application, which is able to access data services on the Web and host device.

This work-in-progress paper provides a brief background to interactive television (ITV) and widget applications. A film rating system is being developed in order to evaluate the suitability of widget-based technologies and techniques for developing new television services.

II. Interactive Television and Widgets

The roll-out of next generation ITV services is predominantly restricted by the immense expense and complicated infrastructure required to deliver quality content to a large audience in a personalised way [1]. To achieve true real-time interactivity, a suitable feedback channel is required to primary and third party application services [2]. ITV applications may be developed and hosted by a range of service providers using a common service delivery platform. The possible types of ITV applications include: content and context aware advertising, direct customer feedback and real-time viewer analytics and ratings. Simply recreating a desktop experience on a television adds little value to the consumer and so new strategies are required to take advantage of the vast capabilities of the Web and multimedia services. Many attempts have been made to emulate a browser experience on television, but it is argued that this does not provide a converged service, but rather an alternative interface for interacting with the Web.

A. Widget Channel

An overview of a typical widget architecture is illustrated in Figure 1. A widget engine is responsible for instantiating and running a set of widgets on the delivery platform in addition to providing a means of accessing local and remote services. In doing this, a user is able to interact with a personalised inventory of widgets acquired from a widget gallery or application store.

Widgets have received substantial media attention and are considered to be a popular technology for providing converged services on consumer electronic devices [4]. Intel has dubbed post-digital television incorporating Web applications as *TV 3.0* and their consumer electronics division has partnered up with Yahoo! to develop a widget channel software framework [5]. This strategy attempts to try and find a suitable way of complementing broadcast services with familiar Web-based applications such as weather, news feeds, online photo albums and video blogs. The Yahoo! widget engine is included on Intel's system-on-chip multimedia processors for televisions and set-top boxes [5].

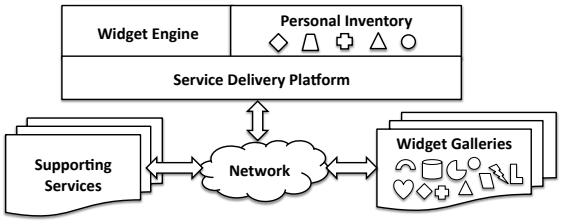


Fig. 1: High-level Widget Architectural Overview

B. Incompatibilities

Although widgets promise to provide a suitable means of delivering cross-platform Internet applications, there is significant fragmentation and incompatibilities between the different widget engines available. The majority of widgets are developed using Web standards, however a widget for one platform cannot run on another without significant modification. Incompatibilities include: packaging, mark-up, event handling, behaviour, configuration data, meta-data, maintenance updates, digital signatures and security models [3].

III. A Film Rating Widget Application

A film rating system is being developed to evaluate a widget based approach to developing an ITV application. The primary use-cases of the system are to allow a user to rate a film they are watching and to browse similar films with good ratings. An architectural overview is provided in Figure 2. The network and content services are abstracted from this research and the core focus is on the film rating widget and its interaction with the platform and application Web service.

A. Development

A widget is not a compiled binary, but rather a single package, which includes any mark-up, styles, configuration, client-side scripting behaviour and supporting multimedia resources. The Yahoo! desktop widget engine is being used to develop the widget as it provides a suitable framework to develop a widget application capable of calling a custom platform API.

Additional components being developed for the film rating system include an application Web service and a platform API. The Web service handles client queries and stores viewer ratings in a central database. The platform API provides an interface for the widget to access a subset of platform functionality, such as the title of the current film being watched.

B. Interfacing with the Platform

Rather than using an engine-specific API, a custom API is being developed to model a generic interface between the engine and underlying platform. Widgets are able to interact with the underlying operating system through a set of API's provided by the widget engine. These API's differ significantly between different engines and so various standardisation efforts are being carried out in an attempt to reduce these incompatibilities. A noteworthy effort is the Open Mobile Terminal Platform's (OMTP) Bondi specifications. Bondi is concerned with how widgets interact with mobile devices. The specifications define a security policy language for widgets and APIs to access platform services such as sending an SMS, making a phone call, taking a picture and accessing the media gallery [6]. Bondi defines 11 APIs and relies on the W3C's widget family of specifications for general functionality [7].

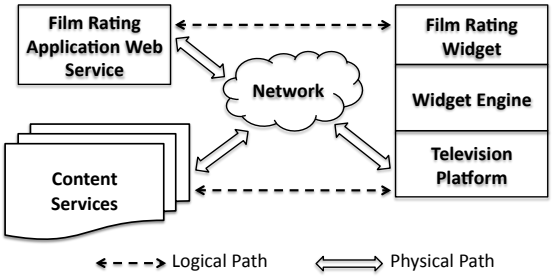


Fig. 2: Film Rating System Architectural Overview

IV. Research Goals

The film rating application use-cases raise fundamental concerns as to how a widget interacts with the platform, content and supporting services. Questions raised from the use-cases include: how should a widget identify a film being rated and how should a widget request that a particular film is played? Answers to these questions depend on whether these actions are carried out by the platform, content service or through a widget's application service.

These kinds of questions are being answered for mobile platforms through initiatives like Bondi, however there is no equivalent effort for widgets on television platforms. The development of a film rating system aims to propose answers to these questions. Further general use-cases will be developed and used to model the relationships between the sub-systems of any widget-based ITV application.

V. Conclusion

Widgets are being used to deliver interactive personalised user experiences on television. One of the challenges of widget-based technologies is the significant fragmentation between different engines and a lack of standard APIs to perform common tasks. A film rating system is being used to develop a platform API to model the relationships between the key sub-systems of a widget-based ITV application.

References

[1] T. S. Perry, "The trials and travails of interactive TV," *IEEE Spectrum*, pp. 22–28, April 1996.

[2] S. Shim and Y.-J. L., "Interactive TV: VoD meets the Internet," *IEEE Computer*, vol. 35, no. 7, pp. 108–109, July 2002.

[3] M. Cáceres and M. Priestley. (2009, apr) Widgets 1.0: Requirements. W3C working draft. [Online]. Available: http://www.w3.org/TR/widgets-reqs/

[4] C. Albrecht. (2009, jan) TVs transform at 2009 CES. [Online]. Available: http://www.businessweek.com/technology/content/jan2009/tc2009019_321332.htm

[5] Intel Consumer Electronics. (2008) Widget channel software framework technology brief: Widget channel: Personalize, enjoy and share your favorite Internet experiences on TV. 320270-003US.pdf. [Online]. Available: http://www.intelconsumerelectronics.com/Download/

[6] Open Mobile Terminal Platform. (2009, jun) BONDI - an open source industry collaboration for widget and web technologies. [Online]. Available: http://bondi.omtp.org

[7] W3C Web Applications Working Group. (2009, jun) Working group wiki. [Online]. Available: http://www.w3.org/2008/webapps/wiki/Main_Page

Paco Mendes completed a bachelors degree in Electrical Engineering at Wits in 2006 and is pursuing an MSc in Software Engineering.

Barry Dwolatzky is a Professor at Wits. He is the head of the Information Engineering Research Programme at Wits and director of the Joburg Centre for Software Engineering.

C.3 Evaluation of a Widget-based Approach for a Television Film Rating Service

To be submitted.

Evaluation of a Widget-based Approach for a Television Film Rating Service

Paco Mendes
Centre for Telecommunications,
Access and Services
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
paco.mendes@students.wits.ac.za

Barry Dwolatzky
Joburg Centre for Software Engineering
University of the Witwatersrand
Johannesburg, South Africa
barry@jcse.org.za

ABSTRACT

The ability to permanently connect consumer electronic devices to the Internet is changing how people interact with Web-based content, applications and services. A widget is an interactive client-side software application that provides an alternative interface to Web browsers. Widgets have recently gained considerable attention as a cross platform client-side application for accessing Web-based services. A film rating service has been developed using a widget-based approach to evaluate the potential of using widgets to provide interactive Web-based applications on television platforms. The rating service consists of a simplified video-on-demand service, a centralised Web service to manage and store user ratings and a widget running on two test platforms. The widget makes calls to the Web service to rate films, query ratings and obtain personalised film recommendations. The Yahoo Konfabulator engine is used to render the widget and handle user interactions. Widget technologies are undergoing considerable development and substantial work is still required to address the many security issues and incompatibilities that exist for how widgets interact with different widget engines and consumer electronic devices. Further work is required to evaluate the suitability of widgets compared to alternative Rich Internet Application environments.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Software Engineering

1. INTRODUCTION

Personalised services are common on the Web, and numerous examples exist for how users with similar interests can interact and share their experiences. Shared experiences include anything from food to technology, entertainment and travel. The Web is so well suited to providing a platform where users can navigate, discover and interact with only content that matters to them, that it is almost taken fore granted. This is in contrast with the majority of consumer electronic devices, such as the television and mobile phone, where users are constrained by limited interfaces and services.

Various attempts have been made to combine traditional broadcast services with those of interactive distributed Web applications [9, 14, 16]. Although a number of converged networks and services already exist, there is still some uncertainty as to how entertainment services should best be delivered in future and what role the Internet and Web will serve in providing a channel of information to support interactive and personalised user experiences. Widgets are small client side applications developed using Web standards and techniques [2]. Widgets have gained significant attention as a technology that enables the delivery of Rich Internet Applications (RIA) on a range of consumer electronic devices.

The objective of this paper is to provide a brief overview of widgets and to evaluate the suitability of a widget-based approach for an Interactive Television (ITV) rating service. The rest of this paper proceeds as follows: section 2 provides a background and definition of widgets and the state of the technology. Section 3 provides a brief overview of ITV and television based widgets. Section 4 provides an overview of the film rating service developed, and finally, section 5 evaluates some of the main advantages and limitations of widget-based approaches.

2. THE RISE OF WIDGETS

Although the Web browser continues to serve as the primary means of interacting with the Web, a growing need to combine services provided on consumer platforms with a range of Web-based services, has given rise to a new class of application known as a *widget*. Unlike a Web page, which is accessed using a Web browser, a widget is a full-fledged interactive application hosted on the client and developed using Web standards. Widgets typically have simple interfaces, yet they can be used to provide rich personalised user experiences by integrating the capabilities of a host device with those of Web applications. There is a lot of media hype around the use of widgets and how they can be used to pro-

vide access to a wide range of services on multiple platforms [12].

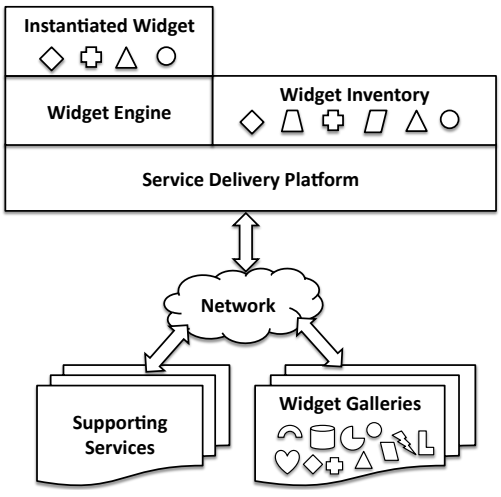
2.1 What is a Widget

Many definitions exists to describe what constitutes *a widget*. The definition used in this paper is based on that used by the World Wide Web Consortium (W3C). The W3C's widget family of specifications define a widget as a packaged, interactive client-side application developed using Web standards and techniques [2]. Unlike a Web page, a widget is hosted on a client and able to access data services both on the Web *and* host device. Widgets require a *widget engine* to render the user interface, handle user interaction and provide a programmatic means of accessing data services both on the Web and host device. A generic overview of a typical widget architecture is illustrated in figure 1. A user is able to acquire a number of widgets from widget galleries or application stores to build their own widget inventories.

2.2 Widget Engine

The widget engine is the application responsible for instantiating and running a set of widgets in a user's widget inventory. The widget engine provides a runtime environment whereby a widget is able to interact with the underlying service delivery platform and network. This runtime environment serves as a layer decoupling a widget from the host platform. The widget engine provides widgets with a means of accessing device specific capabilities and resources through a set of Application Programming Interfaces (API's) and configuration files. In many aspects, widget engines serve a similar role to that of Web browsers and many are built directly on top of Web browser frameworks to incorporate functionality such as rendering HTML mark-up and interpreting client-side scripting [2]. A key difference is the security model which may be configured to allow a widget to access platform specific features [1].

(a) Instantiated Widgets Running on Different Service Delivery Platforms



(b) Architectural Overview of Widget Engine, Service Delivery Platform, Network, Gallery and Supporting Services

Figure 1: Widget Architecture Overview

2.3 State of Widgets

The rapid rise of widget-based approaches has lead to a number of implementations and definitions as to what constitutes a widget and what role it serves. A number of commercial widget engines exist for different operating systems on desktops, mobile phones and media platforms. Different implementations have taken different approaches as to how widgets are realised, and so a number of incompatibilities exist in the widget landscape [2]. The majority of widgets are developed using Web standards, however a widget for one platform cannot run on another without significant modification. Incompatibilities include: packaging, mark-up, event handling, behaviour, configuration data, meta-data, maintenance updates, digital signatures and security models [1]. A significant limitation of widgets is that a user cannot run a widget developed for one widget engine on another widget engine without significant modification to either the widget or the widget engine [11]. These incompatibilities have the potential of restricting widgets from becoming globally ubiquitous and so, numerous efforts are being carried out in an attempt to standardise various aspects of widgets.

In 2005, the World Wide Web Consortium (W3C) identified widget-like technologies, which they initially called "Rich Web clients", as relevant to the future use of the Web and subsequently formed the Web Applications Formats Working Group [20]. The W3C is developing the Widget Family of Specifications in collaboration with a number of partners to attempt to standardise common concerns of different widget implementations. Further efforts are being carried out to try and standardise additional concerns. One such initiative is the Open Mobile Terminal Platform's BONDI specifications, which aims to standardise mobile phone Javascript API's [13].

3. INTERACTIVE TELEVISION

Interactive Television (ITV) is a form of television where the viewer is capable of interacting with media based on their personal choices and physical interactions. ITV has a long and extensive history of approaches to create interactive experiences on television platforms. As discussed by Jensen [9], the evolution of ITV over the last 50 years is characterised by 6 distinct phases plagued by a number of failures. Failures have resulted from a number of reasons, which include the use of premature technologies, expensive infrastructures and poor consumer adoption [14].

The Internet and Web has fundamentally changed how consumers interact with content and services. In contrast to the major failures experienced by ITV, the Internet and Web has experienced rapid growth. The Web has disrupted a number of technologies, including previous attempts to deliver ITV. A more recent attempt to relaunch ITV, has been the convergence broadcast services with services available on the Internet. Early attempts include providing users with services normally accessed on a personal computer, such as:

email, search and chat [9]. Many of these attempts *emulate a desktop* experience on television and it is argued that this does not provide a new seamless service, but rather an alternative browser interface. Simply recreating a desktop environment on a television adds limited value to users and so new strategies are required to take advantage of the vast capabilities of the Internet and Web.

3.1 Widgets for Television

A dominant design for providing Web-based services on television has not yet emerged. In recognition of the poor adoption, a prominent development that has emerged is to use widgets. A noteworthy effort is the *Widget Channel* being developed by Intel and Yahoo [7]. The Widget Channel is a software framework that allows widget applications to run on Intel consumer electronics multimedia platforms. The framework may be included on television sets or set-top-boxes by original equipment manufacturers and it aims to provide a seamless Web experience on television. Lessons learnt from earlier ITV ventures suggest the importance of simple user interactions suited to a television environment [9]. Intel has adopted this by only using a traditional remote so that the platform handles limited user behaviour.

The Widget Channel Software Development Kit (SDK) has been made available to third party developers as of July 2009 [8]. This is not unusual in Web communities, but in contrast to the very closed and proprietary television and set-top-box environments. It is impossible for a single service provider to supply sufficient applications to cater for all niche markets and user preferences, and so the approach taken in launching the Widget Channel shows a change in strategy that is much closer to that normally used on the Web. Widgets present a new opportunity to improve the approach to developing converged Internet-based services for television and other consumer electronic devices. Widget technologies are still undergoing substantial development and so there remain various questions as to how widgets and widget engines should be used in television environments.

4. A FILM RATING SERVICE

A film rating service (FrameRate) has been developed to evaluate a widget-based approach to develop an interactive Web-based application for television. FrameRate is a proof-of-concep film rating service that allows users to rate films they are watching and to browse recommended films based on their rating history. A goal of developing the service is to evaluate how an application typical in a Web environment can be deployed for an interactive media platform through the use of a widget. The application models the key interactions between the widget, delivery platform, media content and supporting Web services.

4.1 Specifications

A set of requirements and functional specifications exist to identify current and potential future features of the service. The three core requirements identified for the service are that FrameRate allows users to:

1. Rate a film during playback.
2. Browse recommended films for a particular genera.
3. Select and play a film on the host platform.

Further requirements defined for the system have not been implemented, but have been used to identify desirable future features that should be considered in the current system design. These include requirements for alternative interface modes, content sources and smarter recommendation results.

A set of use cases have been developed using techniques proposed by Cockburn [3]. The use cases describe what the service needs to do to allow users to *rate*, *browse* and *play* a film on the platform. A key outcome of the use cases developed is the role of a user as an actor on the system. In the IT, Mobile and Web domains a user is normally characterised by a personal account or profile, that may form part of a group of users. Television platforms differ from this in that interaction with the platform is not limited to single users or entire groups. The use cases highlight that primary actors on the system can be a single user *or a group of users* that may rate and browse films based on their collective preferences. A group is dynamic in the sense that a user does not permanently belong to any group. As an example, a group of friends may get together and collectively interact with an application to watch a film best suited to the group. The interaction with the platform should not be restricted to single user profiles but rather based on user presence.

4.2 Assumptions

Certain assumptions have been made to limit the scope of the rating service. Content being rated is restricted to a set of feature films. In practice, content could consist of a range of media including broadcast content, music videos, television series and user generated content. A further assumption made is that the widget will operate in a single window mode overlaying the film.

4.3 Architecture

The FrameRate architecture is based on the generic widget architecture illustrated in figure 1. The notion of a widget gallery and user inventory has been removed, as the widget is already made available on the platform. An architectural overview of the service is illustrated in figure 2. The service consists of three key components:

1. Service Delivery Platform
2. Film Rating Web Service
3. Content Service

4.3.1 Service Delivery Platform

The service delivery platform is responsible for hosting the media player, widget engine and widget application. The widget engine provides a layer of abstraction that decouples the widget from the media player and delivery platform. The role of the widget is to provide an interface for a user to interact with the platform and Web application in a seamless way. The widget serves as a front-end to the Web service, but is standalone in that it does not need to be connected to the service in order for it to instantiate and run on the platform.

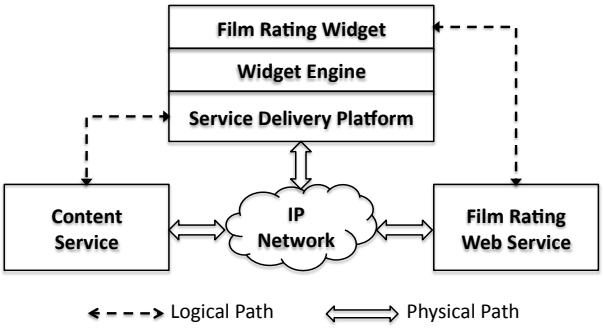


Figure 2: FrameRate Architectural Overview

4.3.2 *Rating Web Service*

A centralised Web service is responsible for storing and managing film ratings for all users on any platform. The Web service is also responsible for generating film recommendations for any clients.

4.3.3 *Content Service*

A content service is required to provide an emulate a Video-On-Demand (VOD) environment for a television platform. The responsibility of the VOD service is to provide any platform connecting to it with access to a remote collection of films. Transport and storage implementation details are not a core concern of the rating service and so only a simplified service is required.

4.4 Implementation

The FrameRate service has been developed and deployed in the Convergence Laboratory at the University of the Witwatersrand's Centre for Telecommunications, Access and Services (CeTAS) [15]. The CeTAS laboratory provides a configurable Next Generation IP Network (NGN) environment where the rating service can be deployed, configured and tested for different platforms.

4.4.1 *Service Delivery Platform*

A desktop computer is being used to emulate a television environment. The Intel-Yahoo Widget Channel engine has not been used as certain features (such as user profiles) are enforced by the Widget Channel implementation. It has been decided to rather use the more generic Yahoo Konfabulator engine as it provides a greater degree of flexibility to model the various subsystems and interactions [21]. Konfabulator is currently only supported for Windows and OS X.

The user interface (UI) is defined in eXtensible Markup Language (XML) and composed of layered Portable Network Graphic (PNG) images of varied opacities. A screen shot of the widget is illustrated in figure 3. The engine is configured to render the widget over the video streams.

The Mozilla SpiderMonkey Javascript interpreter is incorporated within the Konfabulator engine [21]. A native Javascript API offered by Konfabulator, provides a range of functionality that includes timers, event handlers, manipulation of UI elements and access to the XMLHttpRequest object to perform HTTP requests [21]. Timers are used to poll the platform to monitor for any changes and event handlers handle user actions. The engine allows any element described in the user interface XML file to be accessed and manipulated directly by Javascript. A key difference to browser based Web applications is that there is no need to navigate a Document Object Model (DOM) tree as is normally done for HTML based Web pages.

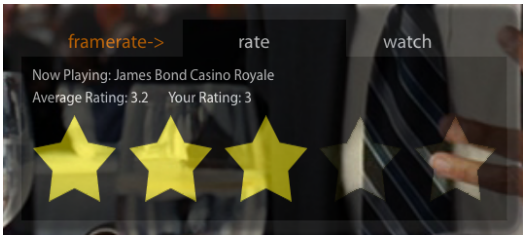
The client platform includes a customised instance of the VideoLan player (VLC) capable of streaming media from the VOD service. Each media stream is tagged by a unique Media ID (MID). The widget application is abstracted from the VOD service and only interacts with the platform using a MID. The platform is capable of requesting and playing any media resource hosted by the VOD service using a Media Resource Location (MRL). The MRL is composed of the MID and path to the VOD service.

Konfabulator allows a widget to interact with the platform using either: a set of native Javascript API calls; HTTP requests to the local platform; shell commands; Applescript in OS X or the Component Object Model (COM) in Windows. Interaction with the media player has been implemented using HTTP requests to the local platform. Macro scripts have been developed for the VLC HTTP interface to allow the widget to acquiring meta-data and control playback for the VLC player [19]. A custom API has been developed to allow the widget to interact with the platform using Javascript. This API abstracts the widget from the underlying platform and the following simplified methods may be used by the widget:

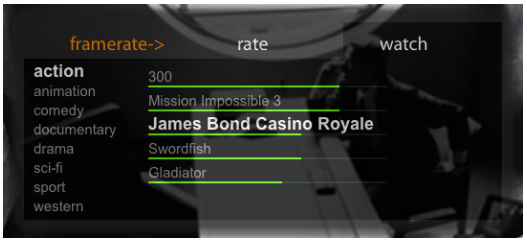
```
void TV.playFilm(MID)

MID TV.getTitle()
```

4.4.2 *Rating Web Service*

The rating Web service has been developed using the Google AppEngine Web application framework [5]. AppEngine pro-



(a) Rate Mode



(b) Browse Mode

Figure 3: FrameRate Widget Screenshots

vides a hosted environment to develop and deploy a Python based Web application capable of handling remote procedure calls (RPC). Web methods allow clients to acquire film meta-data, rate a film for a number of users and get film recommendations for a particular genera. Parameters are passed within the URL of the request and result sets are returned as JSON encoded strings [10]. A simplified representation of the Web methods exposed by the service include:

```
void TestOnline()

json GetFilm(mid)

void RateFilm(user_ids, mid, rating)

json GetGeneraFilms(user_ids,genera,rated)
```

An example response for a single user request is shown here:

```
GET
http://frame-rate.appspot.com/rpc?action=
GetGeneraFilms&id=testu1&genera=comedy&rated=false

HTTP/1.1 200
{"films" : [
  {"mid": "NapoleonDynamite",
    "title": "Napolean Dynamite",
    "description": "...",
    "genera": "comedy",
    "average_rating": "3.8"
  },{"mid": ...},{...},{...}
]}
```

The rating service is responsible for storing all film ratings and test meta-data. A simplified representation of the internal data model is illustrated in figure 4. A resulting outcome of the service is a need for a unique key to identify all film entities for the rating service and VOD service. This *mid* key correlates to the MID used for the VOD service. The widget acquires a film meta-data using the rating service as this removes the need for the widget to be aware of the VOD service. It is preferable that the rating service does not host a meta-data library, but rather dynamically acquires meta-data from a third party source, such as the Internet Movie Database (IMDB) [6].

4.4.3 Content Service

The VOD service provides the platform with access to a collection of films. The VOD hosts films on an Apache Web server and streams films to the delivery platforms over HTTP. VLC has native support for consuming and playing HTTP video streams. HTTP is not optimal for streaming video and controlling playback, but is sufficient to provide a simplified streaming service to demonstrate the interaction of two platforms with remote content streams. A more appropriate protocol is the Real Time Streaming Protocol (RTSP).

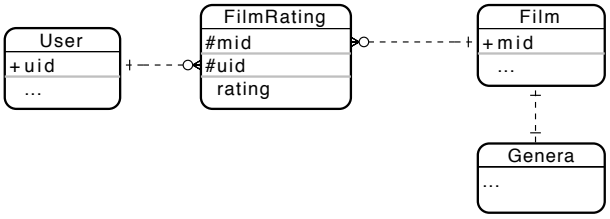


Figure 4: Simplified FrameRate Rating Service Data Model

4.5 Testing

The testing environment consists of two client platforms and VOD virtual machine hosted in the Convergence Lab. The widget platform has been deployed on two client platforms running on OS X and Windows. The Web service is deployed in the Google AppEngine cloud and test data has been generated for 35 feature films and 15 users.

A number of test cases have been identified to test the client widget and Web service based on the use cases developed. As the VOD service is not central to the investigation, no formalised testing has been carried out for this service.

Test scripts have been developed to simulate and test typical activity flows of the Web service. Tests include adding a new film and validating that the JSON representations are correct. The script also allows the creation of randomised ratings for users and films. Integration tests are carried out manually following a set of test cases identified. Sample test cases include:

- Single User Rates Rated Film
- Multiple Users Rate Both Films that have or have not been Rated
- User Rates Non Existent Film

Further work is required to automate testing activities. Automated testing frameworks are available for performing Javascript tests within a Web browser and so further work is required to evaluate the ability of performing automated testing within the Konfabulator environment.

5. FINDINGS

Following the development of the FrameRate service, a number of advantages and limitations of widget-based approaches have been identified. Advantages and limitation exist for both widget-based approaches in general and the Konfabulator implementation. A few of the key findings are highlighted here.

5.1 Advantages

One of the foremost advantages of developing a widget for the Konfabulator engine is the ease of development compared to developing applications for the Web browser. Konfabulator provides a rich environment to develop widgets ranging in complexity. The native API provides an extensive library of functionality that assists in streamlining the amount of Javascript code required to perform common tasks [21]. The native API also reduces the need to include additional libraries to perform common tasks such as parsing XML. Javascript code is significantly simplified in comparison to the equivalent code required to develop a similar

application for a Web browser. A key reason for this is the ability to access and manipulate UI elements directly without having to navigate the DOM tree. The UI XML definition is well decoupled from the Javascript code and the layout can easily be altered without having to alter any of the underlying Javascript code. A further advantage is the ease in which asynchronous requests can be made to the rating Web service in comparison to the more traditional Asynchronous Javascript And XML (AJAX) approaches in Web browsers.

Through the development of a custom API, a further advantage of a widget-based approach is the ability to abstract widget from underlying platform. The widget application is capable of interacting with the underlying platform completely unaware of the VOD service using a very simple Javascript interface.

5.2 Limitations

Widgets allow Web application to access underlying platform features. This is in contrast with the very restricted sandbox environment used by Web browsers [18]. Although access to the underlying platform allows for the development of rich interactive client side applications, a number of security concerns are introduced. It is difficult to ensure that malicious code is not injected into a widget during runtime. As a widget may be able to access a shell on the platform and make HTTP requests to third party domains, this raises significant security concerns. The use of digital signatures alone may not be sufficient to ensure that malicious code is not executed on a client.

Unlike a website hosted on a central server, a widget is hosted on a client and so maintenance updates are required to upgrade distributed widgets to new versions. This introduces new challenges for ensuring that all clients are running the latest stable version.

Limitations of the Javascript language are inherited by the Konfabulator implementation. A significant limitation of Javascript is the ability to apply object oriented programming techniques [4]. As such, code lacks intuitive interfaces to APIs and objects. A further limitation is the testing techniques that can be applied. No unit tests have been developed for the widget client, although this needs further evaluation.

A large portion of the code developed for the widget depends on Konfabulator conventions and the native API. The implementation is loosely coupled from the Web service and media player, but tightly coupled to the engine implementation. Very little functionality is portable to other widget engines even if they also use similar Web standards.

Although the Konfabulator engine supports a subset of Cascading Style Sheet (CSS) syntax, it does not support the inclusion of style hierarchies in a separate file to the layout and scripting. This results in repeated styles for similar UI elements.

5.3 Outcome

Widgets present an opportunity to deliver rich experiences on television using a range of Web-based services. The widget engine provides a useful way of abstracting the widget from the underlying platform and remote Web services. The main advantage offered by widget-based approaches is the ease of development for Web developers used to working in Javascript and XML environments. Although the majority of widget engines make use of Web standards, there are still a number of incompatibilities between different implementations. The significant incompatibilities across the widget landscape limit the portability of widgets across different platforms. Developers are required to choose a particular platform to develop a widget for as it is currently impossible to use the same widget on a number of engines. Standardisation activities should assist in reducing the incompatibilities, however, until a dominant cross platform widget engine emerges, these incompatibilities will continue to restrict the adoption of widget-based approaches.

Widgets provide an attractive means of developing and deploying Web applications on a client as they provide a Web application with the ability to interact with the platform in ways that are restricted by Web browsers. Widgets introduce a number of concerns, however, which including issues already prevalent in the Web browsers. Security and fragmentation in various implementations, continues to remain a key concern of widget-based technologies. Developing client applications using Web standards may suit Web developers, however, no significant benefits have been identified for developing a widget over other RIA frameworks such as JavaFX [17]. Further evaluation of alternative RIA frameworks is required.

6. CONCLUSION

Interactive television has gone through a number of failed attempts, with more recent efforts attempting to converge broadcast and Internet-based services. There is a lot of hype around the use of widgets to converge traditional broadcast services with those available on the Web. Widgets can be used to develop rich client-side applications capable of interacting with the underlying service delivery platform. A film rating service has been developed to demonstrate how widgets can be used to provide a seamless ITV interface to a user. A number of limitations exist for adopting widget-based approaches, which include security concerns and incompatibilities between different implementations. The future of widget technologies and their role as a packaged client-side Web application will largely depend on the outcomes of various standardisation efforts under way. Widgets may provide an easy means of developing client side Web applications, however widgets have not reached a level of maturity where this can be done across multiple platforms in a uniform, secure way.

7. REFERENCES

[1] M. Cáceres. Widgets 1.0: The widget landscape (Q1 2008) W3C working draft. `http://www.w3.org/TR/widgets-land`, apr 2008.

[2] M. Cáceres and M. Priestley. Widgets 1.0: Requirements. W3C working draft. `http://www.w3.org/TR/widgets-reqs/`, apr 2009.

[3] A. Cockburn. *Writing Effective Use Cases (The Agile Software Development Series)*. Addison-Wesley Professional, 2000.

[4] D. Flanagan. *JavaScript: The Definitive Guide, 4th Edition*. O'Reilly, 4th edition, nov 2001.

[5] Google. AppEngine. `http://appengine.google.com/`, sep 2009.

[6] IMDB. The Internet movie database. `http://www.imdb.com/`, sep 2009.

[7] Intel Consumer Electronics. Widget channel software framework technology brief: Widget channel: Personalize, enjoy and share your favorite Internet experiences on TV. `http://www.intelconsumerelectronics.com/Download/320270-003US.pdf`, aug 2008.

[8] Intel Software Network. Widget development kit (WDK) pre-release program. `http://software.intel.com/en-us/articles/widget-development-kit-wdk-pre-release-program/`, sep 2009.

[9] J. F. Jensen. Interactive television - a brief media history. In *EUROITV '08: Proceedings of the 6th European conference on Changing Television Environments, Salzburg, Austria*, pages 1–10, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] JSON. Introducing JSON. `http://www.json.org/`, sep 2009.

[11] C. Kaar. An introduction to widgets with a particular emphasis on mobile widgets. `http://www.symbianresources.com/tutorials/techreports/widgets/kaar07widgets.pdf`, oct 2008.

[12] G. Lawton. These are not your father's widgets. *IEEE Computer*, pages 10–13, jul 2007.

[13] Open Mobile Terminal Platform. BONDI - and open source industry collaboration for widget and web technologies. `http://bondi.omtp.org`, sep 2009.

[14] T. S. Perry. The trials and travails of interactive TV. *IEEE Spectrum*, pages 22–28, apr 1996.

[15] School of Electrical and Information Engineering. Centre for Telecommunications, Access and Services, University of the Witwatersrand. `http://cetas.ac.za`, sep 2009.

[16] S. Shim and Y.-J. L. Interactive TV: VoD meets the Internet. *IEEE Computer*, 35(7):108–109, jul 2002.

[17] Sun Microsystems. JavaFX. `http://javafx.com/`, sep 2009.

[18] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz. Web browser as an application platform. In *SEAA '08. 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 293 – 302, September 2008.

[19] VideoLAN. Building pages for the http interface. `http://wiki.videolan.org/Documentation:Play_HowTo/Building_Pages_for_the_HTTP_Interface`, sep 2009.

[20] W3C Web Application Formats Working Group. Working group charter. `http://www.w3.org/2006/appformats/admin/charter.html`, nov 2005.

[21] Yahoo! Widgets. Konfabulator 4.5 reference manual. `http://manual.widgets.yahoo.com/`, nov 2007.