Quality Control Center for Connected TV

DIMITRI SEMJASJTJENKOV

KTH Computer Science and Communication

Master of Science Thesis
Stockholm, Sweden 2011
Quality Control Center
for Connected TV

DIMITRI SEMJASJTJENKOV

Master’s Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Douglas Wikström
Examiner was Johan Håstad

TRITA-CSC-E 2011:091
ISRN-KTH/CSC/E--11/091--SE
ISSN-1653-5715
Abstract
This thesis investigates techniques used in Web analytics for monitoring Web site activity. The purpose of the investigation is to apply similar techniques to aid organizations monitor and evaluate deployed Connected TV applications. Which in turn may serve as a basis for validating software requirements and verifying that applications meet quality goals. The thesis contributes a method which accommodates monitoring the execution of Connected TV applications in order to measure software and business metrics. The approach requires negligible integration with the applications, yet it still has the capability of measuring numerous metrics beyond what is commonly tracked in Web analytics. This paper includes the specifications and implementation details of an operational prototype system which utilizes the suggested approach. The system not only consists of a novel implementation of the page tagging method used in Web analytics to monitor Web pages. It also incorporates the whole data flow from the Connected TV device all the way to the analysts computer screen.

Keywords: IPTV; Connected TV; Smart TV; TV apps; Web analytics; quality control; business intelligence.

Referat
Kvalitetskontrollcenter för Connected TV
Preface

This document is a degree paper authored by a student at the Royal Institute of Technology in Stockholm, Sweden for his Master of Science in Computer Engineering degree. The paper is based on a study performed by the author from January to April 2011. The study was supervised by Ted Björling at Accedo Broadband AB and took place at the company’s Stockholm office.

Acknowledgment

The author wishes to thank the people at Accedo Broadband AB for their part in this work, in particular Ted Björling without whom this thesis would not have been completed. In addition, the author wishes to extend his gratitude to Douglas Wikström at KTH for his invaluable input during the writing process. I also wish to thank family and friends who have been an important source of motivation throughout this study and events leading up to it.
Glossary and abbreviations

**ActionScript** – Script language used by Flash applications.

**AV** – Audio-visual, work with both sound and video component.

**Connected TV** – Occasionally Smart TV. See IPTV.

**DOM** – Convention for accessing and manipulating HTML documents.

**Ethernet** – Set of specifications and standards for local area networks.

**HTML** – HyperText Markup Language, markup language used for most Web pages.

**HTTP** – Hypertext Transport Protocol, network communication protocol used by the WWW.

**IPTV** – Internet Protocol television, network enabled television platform.

**JavaScript** – Script language executed client-side to create dynamic Web pages.

**JDBC** – Java Database Connectivity, Java API for accessing a database server.

**JSON** – JavaScript Object Notation, JavaScript code for human readable data exchange.

**MAC address** – Media Access Control, unique identifier assigned to network interfaces.

**Query string** – Set of key/value pairs used to transmit simple data by appending it to a URL.

**URL** – Uniform Resource Locator, defines the location and type of a resource.

**W3C** – World Wide Web consortium, community of organizations developing Web standards.

**WWW** – World Wide Web, part of the Internet.
Content

1 Introduction.................................................................................................................. 1
  1.1 Context.................................................................................................................. 1
      1.1.1 Quality control......................................................................................... 1
      1.1.2 Software maintenance........................................................................... 1
      1.1.3 Web development.................................................................................... 2
      1.1.4 Internet-based TV.................................................................................... 4
  1.2 Purpose.................................................................................................................... 5
      1.2.1 Project........................................................................................................ 5
      1.2.2 Monitoring framework............................................................................. 6
      1.2.3 Use case.................................................................................................... 7
  1.3 Obstacles................................................................................................................ 7
  1.4 Requirements and constraints.............................................................................. 9

2 Related work.............................................................................................................. 11
  2.1 Web analytics....................................................................................................... 11
      2.1.1 Measurements......................................................................................... 11
      2.1.2 Data management.................................................................................... 13
      2.1.3 Presentation.............................................................................................. 13
  2.2 Connected TV analytics....................................................................................... 13
  2.3 IPTV quality of service......................................................................................... 14
  2.4 Data models.......................................................................................................... 15
  2.5 Scalability............................................................................................................. 17
      2.5.1 Database................................................................................................. 17
      2.5.2 Server....................................................................................................... 18

3 Implementation........................................................................................................ 20
  3.1 System model....................................................................................................... 20
  3.2 Monitoring tool..................................................................................................... 21
      3.2.1 Deployment............................................................................................... 22
      3.2.2 Monitoring............................................................................................... 23
      3.2.3 Reporting................................................................................................. 25
  3.3 Presentation.......................................................................................................... 25
      3.3.1 Scripts....................................................................................................... 26
      3.3.2 Graph package.......................................................................................... 26
  3.4 Database............................................................................................................... 27
  3.5 Web server........................................................................................................... 28
      3.5.1 Document server..................................................................................... 28
      3.5.2 Database interface................................................................................... 30

4 System review.......................................................................................................... 31
  4.1 Requirements....................................................................................................... 31
  4.2 Monitoring tool..................................................................................................... 32
      4.2.1 Metric list................................................................................................. 32
      4.2.2 Maintainability......................................................................................... 33
      4.2.3 Limitations............................................................................................... 33
  4.3 Web server........................................................................................................... 34
      4.3.1 Scalability................................................................................................. 34
      4.3.2 Limitations............................................................................................... 35
4.4 Database........................................................................................................................35

5 Conclusions and recommendations .............................................................................36
5.1 Contributions...............................................................................................................36
5.2 Next steps ..................................................................................................................36

Bibliography ..................................................................................................................39

Appendices
A Integration manual ........................................................................................................41
B Screen capture ...............................................................................................................42
C Reporting interface ......................................................................................................43
D Data interface ...............................................................................................................45
E Metric list ....................................................................................................................46
1 Introduction

Assuring applications fulfill quality requirements is a vital aspect in any software product and that is no different for Web applications targeting the emerging Connected TV platform. These Web applications are deployed onto remote devices which generally are not under the control of application developers. Quality control of the Web application therefore has to rely on procedures able to overcome the remote nature of the operational environment of these applications. One solution to this is a tool able to deliver information on various aspects of the Web application execution. In order for such a tool to be of practical use it also needs a supporting framework which it operates within. This thesis describes a system incorporating such a tool and supporting framework.

With the intention to familiarize the reader with the subject of this thesis the following chapter introduces several pertinent concepts. The next couple of sections briefly describe the context and purpose of the project which this thesis is based on. After which follows a summary of the obstacles which exist with regard to the purpose of the project and what limitations were placed on the project. In the remainder of this paper the above mentioned project is referred to as “the thesis project”.

1.1 Context

In order to familiarize the reader with concepts pertaining to the subject of this thesis the following sections present several of them.

1.1.1 Quality control

Quality control in software development consists of the tasks which aim to ensure that project quality goals and standards are fulfilled. Quality control, which can be traced back to manufacturing industries, has influenced many concepts in software engineering and is one of the underlying principles of software testing. In practice quality control is commonly implemented by inspection of the software at different stages of development. Information gathered during inspections is then used to evaluate the software with respect to project goals and requirements. Sommerville (2007) recommends incorporating quality management procedures prior to and throughout the software development cycle.

1.1.2 Software maintenance

Software maintenance is a process within software engineering which may benefit from implementing quality control procedures. Software maintenance takes place after a product has been delivered and can be pictured as an evolutionary development process. The by far most common tasks during maintenance is implementing changed or new user requirements
or adapting to changes in the operational environment (Lientz & Swanson 1980). Quality control procedures become vital when implementing new or changed requirements during maintenance to verify that product quality is maintained.

One way to incorporate quality control during maintenance is to perform internal software testing before delivering an updated product. This is the same approach commonly employed during validation and verification processes. A supplemental approach is to verify the software after deploying updates into the operational environment. The latter approach should not be confused with the ship-now-fix-later mentality frowned upon by software developers at large. Instead one should look at it as another form of quality control which can potentially be more accurate than simulated tests.

**Metrics and monitoring**

To accommodate evaluation of a software product a set of metrics needs to be specified. The purpose of these metrics is to allow objective software characterization with regard to project goals and software requirements. Without defined metrics two individual evaluations of the same software product may result in two correct yet disparate conclusions, an example of this follows. If there was no temperature metric such as Celsius something as simple as the boiling point of water would be ambiguous depending on who you asked. One claim could be; *heating the water on a stove for a period of time*; and another; *when the water evaporates*.

Monitoring tools is one way of acquiring measurements of the metrics. There exist many kinds of monitoring tools in scientific and technological fields of study but this thesis only covers the subset of software monitoring tools. Note that the software part refers to what is being monitored, the tool itself may or may not be implemented through software. For the remainder of this paper the software part may be dropped when referring to software monitoring tools. In software development, monitoring tools are often custom built for a particular project or task but there exist many commercial off the shelf tools. The choice whether to go with custom solution or commercial product is usually decided by factors such as feature set, supported hardware and cost. The main purpose of the monitoring tools is to observe and measure certain metrics of a software system. Performance metrics such as memory usage and code related metrics such as lines of code can be of interest during software implementation. On the other hand business centric metrics such as number of concurrent users may be of greater importance during maintenance of multi-user applications.

### 1.1.3 Web development

Web development is a relatively new field within software development and encompasses work surrounding Web site development. A Web site is a collection of Web pages and can serve a number of purposes; electronic shops, social networks or in a broader definition, any application which is accessed through the Internet or a local area network.

Web sites in general possess several characteristics which make them potentially difficult to evaluate without deploying them in their operational environment. This is often the case with applications which operate over the Internet since recreating the Internet in a testing environment is nigh to impossible. In practice, software is used to simulate the behavior of Internet traffic but this in turn may introduce irrelevant factors due to errors in simulation models. This lack of a realistic testing environment often leads to Web applications exhibiting unforeseen behavior when they are deployed. Developers combat this by utilizing the
maintenance process to rectify software errors and monitoring tools can greatly aid with the initial identification of errors.

There exist a large number of software monitoring tools for Web applications which together cover numerous metrics. These monitoring tools are not solely used in software development processes but also within business intelligence. In the latter case they are used to evaluate business aspects of the application and therefore rely on different metrics. Nonetheless, the main concepts of monitoring tools remain more or less the same independent of the intended use.

**Internet applications**

A subset of Web applications are Internet applications, the majority of Internet applications take the form of Web pages viewed through Web browsers. Web pages are often static documents which present resources such as text, graphics and sound. It is also increasingly common to find so called dynamic Web pages which adjust their content based on various parameters and actions of individual users. This section introduces several key concepts pertaining to developing Web pages.

Internet applications in general and Web pages in particular, establish a client/server relationship between user and the application provider. Clients send requests to a server which are then processed using predefined logic and some action is taken according to this logic. For Web pages this action is commonly a formal response containing some data sent back to the client. When developing Internet applications for browsers there are a few notable restrictions compared to Web development in general. Browsers are limited to executing code written in script languages such as JavaScript and mark-up languages such as HTML. These languages have severely limited access to hardware and the operating system to protect the user from malicious programs. This has the unfortunate consequence of restricting legitimate applications. Programming language restrictions do, however, not exist on the server side and it is possible to implement servers using for instance Java, C and its extensions.

Another key factor when developing Web pages is the use of standards and recommendations. The most prominent of these are the document object model (DOM) recommendations from World Wide Web (W3C ) consortium. These particular recommendations are so widely endorsed that they are often referred to as Web standards. W3C DOM defines a set of guidelines with the intention to standardized manipulation and presentation of individual Web pages elements. These recommendations allows scripts and external modules to manipulate page elements in a common way to produce dynamic Web pages.

**Event driven**

One section of the W3C DOM recommendation is concerned with events specification. These events enable scripts to execute instructions in accordance to triggered events instead of a predefined order. Events can be triggered by numerous user actions and software processes and the following list contains a small selection of events which a Web page can trigger:

- **onClick** – User clicks a Web page element such as a button or a link.
- **onLoad** – All elements of the Web page are fully loaded.
- **onMouseOver** – Mouse pointer is moved over a Web page element.
HTML elements of a Web page each have several events associated with them and events are triggered individually for each element. For instance, clicking a button only triggers the event for that button and not for all buttons contained in the Web page. Incorporating events in a Web page is a straightforward process. An event handler is registered to an event and the handler is then called whenever the event is triggered. Event handlers are functions containing instructions which are to be executed when the event is triggered. The procedure for registering an event handler with a particular event is performed by the assignment operator. Below is an example of such an assignment which registers the function `eventHandler` with the click event of a button.

```javascript
buttonXYZ.onClick = eventHandler;
```

Initially this was the only way to register an event handler but later recommendations from W3C introduce the concept of event listeners. Each event may have several such listeners and below is an example of how a listener can be used to achieve the same result as the above example.

```javascript
buttonXYZ.addEventListener('click', eventHandler, false);
```

The first and second parameters directly correspond with `onClick` and `eventHandler` in the first example. The third parameter specifies whether the event is captured and can be ignored within the scope of this paper. The first example represents what is referred to in Web development as DOM level 0 events while the latter example represents a DOM level 2 event. Despite the obvious benefits with level 2 events both types remain in use even in newly developed Web pages.

### 1.1.4 Internet-based TV

With the ongoing introduction of broadband connections in households an increase in demand for on-demand TV content is to be expected. IPTV, and more recently Connected TV, are in response to this demand and are quickly gaining ground lost by traditional TV broadcasts. IPTV and Connected TV are two very similar technologies which allow distribution of TV centric home entertainment via packet switching networks such as the Internet. They are by no means the only technologies for delivering TV over packet switched networks but as the title of this thesis suggests are in focus.

Traditional TV, also referred to as linear broadcasts, do not allow user feedback apart from adjusting TV settings such as volume. The new technologies allow more intricate user interaction and this is evident by the services which are now being offered on the new platforms. Before delving deeper into Internet-based TV it is useful to understand the difference between IPTV and Connected TV. The key difference is that Connected TV uses the Internet to deliver content to end-users while IPTV employs so called managed networks for the same task.

Managed networks are, unlike the Internet, monitored and maintained by one entity, usually an Internet Service Provider (ISP) or telecommunications operator (telco). The are several benefits with delivering TV content over managed networks compared to the Internet (Goldberg & Kernen 2007). Foremost, it allows service providers to guarantee quality of service (QoS) to IPTV end-users. When a service is delivered over open networks maintained by several operators it becomes increasingly difficult to guarantee a high level of QoS. As household broadband connections and the Internet infrastructure improves the benefits of managed networks decreases. It is today possible to provide consumer grade QoS video...
content over the Internet and this is reflected by the recent boom in Connected TV sales (DisplaySearch 2010).

Although there is some confusion as to the definition of IPTV this document takes the view presented in (MOCA 2008). The referenced paper is by no means required reading and simply being aware of the main difference with Connected TV suffices.

**TV apps**

Traditionally TV content has been a linear experience, live broadcasts are a good example of this. The limitation stems from the technology historically employed to deliver these broadcasts. Service providers have had to resort to other channels such as telephony to allow user interaction. Internet-based TV on the other hand allows users to directly interact with the TV content, basically transforming the viewer into a user. Because Internet-based TV is still a relatively new technology content providers are still exploring various forms of interactivity. Currently the trend is shifting towards so called TV apps which at their core are Internet applications targeted towards owners of Internet enabled TVs. TV apps have a lot in common with Web pages which are normally browsed on a computer but are tailored for the living room environment. The following list contains a few examples of what types of TV apps currently exist:

- Single player mini-games.
- Weather forecasts and other news services.
- Adapted versions of Web based services such as social networks.
- Information services integrated with live broadcasts.

**API**

In software development it is common to integrate commercial off the shelf or open source software with the system being developed. This can reduce development costs but in order to accomplish this the two software packages need to communicate with each other. An Application Programming Interface (API) accommodates this communication by defining an interface that external systems may employ. Connected TV APIs are of particular interest within the context of this thesis and are referred to as simply “TV APIs”. The APIs allow installed applications to communicate with the operating system of the hosting TV device. TV APIs often resemble Web browser APIs but incorporate additional modules to allow communication with media players and hardware such as network interface.

**1.2 Purpose**

*The following sections introduce the goals of the thesis project.*

**1.2.1 Project**

The initiator of the thesis project is a leading provider of TV apps and content aggregating platforms for IPTV and Connected TV. The company carries a large app portfolio developed in-house. As well as offering development services such as porting and integration of existing applications to IPTV and Connected TV devices. A content aggregating platform is a concept which takes individual content and presents it as a package, for instance a video on-demand
service. Aggregation platforms can be viewed as large TV apps and are therefore referred to as such for the remainder of this paper. The main goal of the project was to develop an unobtrusive cross-device software monitoring tool for TV apps. Let us break this down to get better understanding of this goal. In the previous sections the concepts of monitoring tool and TV app were discussed, let us now look at the unobtrusive and cross-device properties.

**Unobtrusive**

Because the tool will monitor deployed apps it is important that end-users do not notice it. In addition, to avoid inflating development costs of TV apps it should not require time consuming integrations with apps. Thus, the property has different meaning depending on whose viewpoint it refers to. From the end-users viewpoint the property means that the tool does not have a perceivable effect on the users experience of the TV app. Potential adverse effects to the user experience can be but are not limited to, user input lag and animation slowdown. From the development viewpoint unobtrusiveness means that adjustments to accommodate the monitoring tool made to application design and implementation processes are negligible. This can be summarized in the following list:

- **End-user unobtrusive** – no noticeable changes to monitored TV app.
- **Development unobtrusive** – only marginal adjustments of current development processes when integrate monitoring tool.

**Cross-device**

An often vital property of Web applications, especially aimed for the Internet, is the ability of the application to function properly on various browsers and devices. This is made problematic by Web browser companies who often seem to prioritize new functionality over maintaining standards. There is a similar situation with IPTV and Connected TV manufacturers, the main difference is that these platforms lack general standards in the first place. This has led to a number of differences between TV brands which force content providers to spend precious resources trying to support several brands. Requirements on the thesis project call for a monitoring tool which is able to function on a range of TV brands.

**1.2.2 Monitoring framework**

A monitoring tool in itself serves little purpose, there are a number of parts it relies on to be of practical use to developers and other relevant personnel. In the remainder of this document we refer to these supporting parts as “the tool framework” or alternatively “framework” when the context permits. The following list contains supporting functionality the framework should provide:

1. Deployment procedures for the monitoring tool.
2. Tool interface for transmitting monitored metrics.
3. Information management of the transmitted metrics.
4. Presentation to accommodate analysis of metrics.

The remaining part of this section looks at each item in the above list.
Deployment
The monitoring tool may be required to operate in a remote environment which is running the monitored app, in other words, the end-users TV. If this is the case, a deployment procedure is required to handle initial deployment of the tool as well as future updates of the tool.

Interface
If the tool operates on a remote device it will require procedures for transmitting metrics which are being monitored. The monitoring tool will potentially be used to monitor many app instances, the interface will therefore have to cope with many concurrent transmissions.

Information management
Continuously monitoring one or many app instances may generate a large amount of data. In addition, generated data will have to be persisted for the purpose of historical analysis. Thus, it becomes vital to have a strategy for managing large collections of data.

Presentation
The data which will be accumulated in the aforementioned steps will require additional processing before it can be of practical use to a developer or analyst. Thus, the framework needs to accommodate a presentation process.

1.2.3 Use case
The project initiator organization envisions that a TV app monitoring tool can aid developers and also enhance business decisions. Developers can use the tool to better evaluate technical aspects and identify errors in TV apps as well as make more informed design decisions. Business analyst on the other hand may use the tool to make a more informed analysis of TV apps. It is suggested by (Lewis 2009) that business intelligence and software testing may coalesce in the future. Wilson (2010) describes a case study employing Web analytics to improve business-to-business Web site performance. The presence of a second user group can potentially introduce conflicting requirements due to differing levels of computer expertise.

1.3 Obstacles
The thesis project set out to develop a monitoring tool for TV apps and accompanying framework. In order to reach those goals several obstacles had to be dealt with. The following is a list of those obstacles which are thereafter described in more detail:

- Lack of standardized TV APIs.
- Unobtrusiveness
- Metrics
- Scalability

Lack of standardization
Connected TV and IPTV is a rapidly growing market with a steady stream of new TV models
with improved hardware capabilities. Manufacturers are keen on introducing new features from a marketing viewpoint, while, unfortunately, standardization is overlooked in the process. In addition, manufacturers rarely cooperate with each other when it comes to providing consistent APIs. This is a major hindrance for TV app developers because it severely limits portability of TV apps. It is rare indeed that a TV app will work identically on two different TV brands. More often than not the TV app will require extensive code rewriting in order to function properly on each additional brand.

Individual manufacturers have more or less unique TV APIs which may or may not support Web standards. There are manufacturers who comply with existing Web standards by integrating commercial off the shelf (COTS) Web browsers into their products. The standards are, however, not maintained by plug-in modules which extend browsers hardware capabilities. Many times modules from different manufacturers serve the same purpose and have similar functionality but differ on some aspect which is critical for app implementations. Moreover, some manufacturers do not even comply with common Web browser standards such as W3C DOM. The manufacturers instead provide their own solutions which often not only lack the functionality of these standards but are also radically different from Web browsers. TVs employing such radically different solutions require TV apps specifically tailor-made for them.

What does this all mean for a TV app monitoring tool? Clearly, if TV apps require extensive work to function on devices from several manufactures any software targeted for that environment will encounter the same difficulties. This is one of the main hurdles for a monitoring tool which is deployed on IPTV or Connected TV.

Unobtrusive

Developing a tool which monitors TV apps unobtrusively to end-users can be as straightforward as extending the app code with profiling instructions. This is a common practice in app development due to the lack of debugging features on TV devices. With a few adjustments the same principle could be used on deployed TV apps. This approach, however, goes against the unobtrusive property the project is aiming for. The thesis project has the goal of a tool which is unobtrusive to app developers which poses a considerable challenge. One way to avoid this issue altogether is to monitor TV apps remotely by tapping into the communication feed of the app which commonly relies on HTTP. The TV app and tool are then decoupled and developers do not need to take the tool into account when developing apps. Unfortunately this approach can only provide rudimentary metrics compared to a tool which runs on the same device as the app. Because of the severe limitations of this approach it is not considered satisfactory within the context of this document.

Metrics

A monitoring tool accommodates evaluation of TV apps with regard to relevant characteristics such as performance and reliability. The tool achieves this by providing measurements on various aspects of the app execution. These measurements make up the metrics which are vital to software evaluation and in turn to quality control. The decision of which metrics to monitor has a great impact on the whole quality control process. The metrics should have the capability to reflect any development standards in place within the organization. Defining which particular metrics to monitor is not within the scope of this paper or the underlying project. It is, however, vital to the project to define the limitations on what metrics the tool is
able to monitor. This task in itself can be demanding since each individual manufacturer has their own policies which define which aspects of app execution is measurable.

**Scalability**

As IPTV and Connected TV continue to overtake traditional TV as the preferred system for home entertainment, TV apps will gain a larger audience. In order to maintain an accurate evaluation of TV apps a large portion of the app instances should be monitored. Monitoring such a large number of devices may put a heavy load on any central structure of the tool framework such as server or database. The property of managing increased system loads is referred to as the scalability of a system. In order to safeguard against increases in demand, implementations of critical links within the tool framework should be scalable with additional hardware. This requires identifying the critical points and addressing the issue for each of them.

**1.4 Requirements and constraints**

This section presents the key requirements and limitations imposed on the thesis project. It begins by introducing the requirements and then goes on to defining the constraints.

**R1 Monitoring tool**

The primary requirement of the project is the development of an unobtrusive cross-device software monitoring tool for Connected TV apps. This is perhaps more akin to a goal than a requirement, nevertheless it may be viewed as a compound requirement. To clarify the requirement and make it more concrete it is broken down into constituent parts as follows. Unobtrusive and cross-device properties are defined as per section 1.2.1. Software monitoring tool as per section 1.1.2 and finally Connected TV and TV apps as per the definitions in section 1.1.4.

**R2 Framework**

The secondary requirement of the project is the development of a monitoring tool framework as per section 1.2.2. Again this requirement is reminiscent of a goal than anything else but the referenced section gives a sufficient definition of this requirement.

**R3 Software approach**

The monitoring tool of requirement R1 should not rely on specific hardware to be installed at the location of a supported Connected TV device. This requirement originates from the simple fact that installing additional hardware in the users household is deemed infeasible by the project initiator. Content developers are not responsible for the end-user service and thus there are certain legal and logistic problems with installing any such hardware. In addition to this, external hardware devices cannot gain access to the internals of a TV app without modifications to the TV. Internal hardware devices on the other hand would require extensive modifications to the TV which at the least are costly. Note that this requirement does not apply to other parts of the framework which are under the control of content developers.
R4  API restricted
The monitoring tool of requirement R1 should only rely on measuring methods possible within the context of the API provided by the TV manufacturer. It shall neither rely on custom modules such as plug-ins which require firmware updates or any other techniques which require modifications to the TV API. Even though content developers have an open dialogue with Connected TV manufacturers, relying on custom modules for the monitoring tool is susceptible to legal and logistical issues. Operating within the confines of the API guarantees that content developers maintain full control over of the tool implementation. In short, if the tool is operating within the confines of the TV APIs it does not require additional permissions from manufacturers.

R5  JavaScript
Connected TV APIs commonly interface through JavaScript and generally only support execution of JavaScript and in some cases ActionScript. This effectively limits the monitoring tool implementation to either JavaScript or ActionScript which can integrate with JavaScript through a common interface. Therefore, within the context of the thesis project the monitoring tool implementation shall be limited to JavaScript. However, note that this limitation is not imposed on other parts of the framework as described in section 1.2.2.

R6  Metrics
Requirement R4 has the consequence of limiting the set of possible metrics the monitoring tool is able to track and measure. Apart from metrics which can be derived from the functionality provided by the TV API, the tool is allowed to track metrics internal to the monitored TV app. This in part excludes metrics which are measured either on the service providers end or along the network used for delivering the TV app. Note that this last limitation is not intended to exclude metrics acquirable by those means. Instead it aims to limit the scope of the project to one source of measurements.

R7  Device restrictions
There is a steady stream of new home appliance manufacturers entering the Connected TV market. As described in section 1.3 it is non-trivial to implement cross-device applications for these devices. The monitoring tool is therefore limited to devices from three Connected TV manufacturers namely LG, Panasonic and Samsung. This limitation is based on which devices and API documentation were available to the thesis project while still maintaining a diversity among TV APIs.
2 Related work

In this chapter we introduced existing research and applications which are related to this thesis and underlying project.

2.1 Web analytics

The concept of monitoring various aspects of Web pages is an established field within the domain of business intelligence. This field is referred to as Web analytics and serves to aid businesses in the evaluation of their Web presence. There are numerous similarities between Web pages and TV apps and this section therefore begins by looking at Web analytics.

As touched upon previously, the purpose of Web analytics is to allow businesses to evaluate and understand their Web presence. This is achieved by collecting measurements on various aspects of Web usage and subsequent analysis of the accumulated data. Web analytics can thus be broken down into the following three parts:

- Metrics, measurements on Web page usage.
- Data management, maintaining accumulated data on Web page usage.
- Presentation, analysis and reporting of Web page usage.

The observant reader should recognize these concepts from previous sections of this paper. The above metrics part corresponds to the metrics found in section 1.3. Management and presentation respectively correspond with information management and presentation of section 1.2.2.

2.1.1 Measurements

We begin with a few examples of metrics common to Web analytics; number of unique visitors, number of interactions with a particular Web page element, geographical location of visitors. Two techniques to measure and collect metrics are Web server log file analysis and page tagging (Clifton 2010). Figures 1 and 2 depict simplified models of Web server logging and page tagging respectively. Web servers keep a detailed log file of all user transactions, these files can be parsed for relevant data which can later be used for Web page analysis. Page tagging consists of adding scripts to existing Web pages, these scripts are then executed locally on the users computer when visiting the Web page. Employing scripts which report user statistics to a central server allows for evaluation of tagged Web pages.
Tables 1 and 2 contain drawbacks to Web server log file analysis and page tagging respectively, for further details see (Clifton 2010). The listed drawbacks are mostly disjoint and in many practical applications they can be negated by using both methods. When utilizing both methods concurrently the analysis of the collected data has to be especially sound not to arrive at flawed conclusions. For instance, since both methods are able to track the number of visitors, a procedure for counting unique visitors should be defined to avoid counting one visitor twice.

**Table 1. Limitations of Web server log files.**

<table>
<thead>
<tr>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web browser cache hides usage from Web server.</td>
</tr>
<tr>
<td>Proxy servers mask unique users to appear as one.</td>
</tr>
<tr>
<td>Part of the data is caused by Web crawlers.</td>
</tr>
<tr>
<td>Information limited to HTTP requests.</td>
</tr>
</tbody>
</table>
2.1.2 Data management

The ability of Web analytics to accurately derive models and conclusions is dependent on the amount of relevant data which is available during analysis. Gathering too little data may result in flawed conclusion and maintaining large amounts of data requires sophisticated data management tools. The by far most common data management tool is a database and the key question is which database model to employ. A more detailed description of databases and associated models is presented in section 2.4.

2.1.3 Presentation

The final step in a Web analytics framework is the analysis interface. In this step the collected data is transformed into a presentation appropriate for human review. A prominent presentation technique is the use of various graphs which are able to visualize data in a form suitable to analysts. The presentation aspect and with it the data analysis is perhaps the most important aspect of Web analytics. It should therefore come as no surprise that the presentation interface is one of the most evolving aspects of commercial Web analytics frameworks. As mentioned previously, graphs are the most common technique used in Web analytics to visualize underlying data. These range from simple line and pie charts to more advanced network graphs but they all serve the same purpose, to visualize one or several metrics. The user has the ability to adjust the information displayed in each graph by filtering by date and other more intricate properties such as geographical location. The end result is a flexible yet structured interface which can be customized for several purposes.

For readers who wish to delve deeper into practical details this section is concluded with a list of a handful commercial Web analytics providers. For the more academically oriented readers an alternate source of information on practical Web analytics is found in (Phippen et al. 2004).

- Adobe Omniture
- Coremetrics
- Google Analytics
- Webtrends

2.2 Connected TV analytics

Towards the end of the thesis project two companies announced commercial Web analytics services with support for Connected TV. Obviously their products are of great interest to this study since they may potentially offer functionality identical to what the thesis project was aiming for. These upstarts, however, had no effect on the outcome of the thesis project because the project was in its final stage by the time of the announcements. Nevertheless, a brief summary of what these alternative products offer is contained in this section.
Capptain

The first company has been providing Web analytics solutions for several platforms and mobile devices such as Android and iPad. Recently they announced support for Smart TV, which is a Connected TV device manufactured by Samsung. Extensive documentation for the company’s Smart TV support is available on their Web site and this was used for the review of their product. The company offers support for several devices but the review was limited to investigating the Smart TV support. The documentation indicates that so called page tagging, discussed in section 2.1.1, is employed by the company to acquire metrics. A script hosted by the company is included in the HTML documents of a TV app. The script is then invoked when the app files are loaded on the TV and it reports certain information to a central server. The script also provides an API which app developers may use to report additional metrics during execution but this requires integration with the TV app. The company claims that the integration is estimated to take up to five hours for each TV app which is to incorporate the additional tracking. This is a substantial investment for TV apps which often are very small in scope.

The presentation front-end the company provides uses Web browsers for its user interface and presents various graph views of the metrics tracked by the script. To begin tracking a TV app it must first be registered with an account and by doing so receives a unique identifier. This identifier is then used when setting up the page tagging in order to differentiate reports from different apps.

Noble metrics

Unfortunately, the second company did not wish to disclose information pertaining to their service limiting the review to marketing brochures. Therefore comparison with and speculations about their product is omitted from this paper.

2.3 IPTV quality of service

Section 1.1.4 points out that IPTV and Connected TV are similar platforms and therefore may share similarities when it comes to quality control. In addition, because IPTV was introduced to home users over 10 years prior to its Connected TV counterparts, the IPTV platform is more mature. These two factors make IPTV quality of service processes potentially a wealthy source of inspiration for Connected TV quality control. When looking at IPTV service quality there is a recurring focus on audio/visual (AV) quality metrics (Erman & Matthews 2008), (Staelens et al. 2010) and (Latré, De Turck, Dhoedt & Demeester 2007). This is perhaps no surprise as AV broadcasts are still the dominating content on IPTV which is often operated by TV broadcasting specialists such as cable TV operators.

Quality of service (QoS) is a term used when defining the quality of a service transmitted over a network, commonly an AV stream in the case of IPTV. Quality of experience (QoE) is a closely related term but focuses on more subjective aspects, QoS on the other hand is something objectively measurable. QoE is often captured by an individuals subjective opinion and while opinions can be captured they cannot be measured in an objective sense. Because IPTV services are delivered via a managed network, operators are able to measure QoS from end to end and this gives a complete picture of the service quality. Factors such as packet loss and jitter are two metrics used by IPTV service providers to define QoS. To measure what impact metrics have on QoE of an AV stream, jitter and packet loss is induced on a network.
carrying the stream in a testing environment. Thus, a relation between QoS and QoE is established.

### 2.4 Data models

Previous sections have touched on the subject of data management with regard to this thesis and Web analytics in general. This section contains a short survey of several prominent data models employed by databases to store data.

**Relational model**

The most common model in commercial systems is the relational model and it benefits from widespread support and development.

**Disadvantages:**
- Only one value per column is allowed.
- Rows must be unique.
- Values in the same column must be of same type.
- Entries must follow predefined schemes.

**Advantages:**
- Development backed by large investments.
- Proven reliability.
- Backed by mathematical set theory.
- Entries are able to contain several facts.

**EAV model**

EAV is an abbreviation of entity-attribute-value. The EAV model is implemented with tables where each row consists of three predefined columns, the these columns give the models its name. The first column is an entity id, the second an attribute of the entity and the third a value of the attribute.

**Disadvantages:**
- Each row limited to one fact.
- Less efficient with attribute-centered queries.
- No inherent constraint checking e. g. non-null.

**Advantages:**
- Efficient with entity-centered queries.
- One attribute may have different value types.
- Entities can have several attributes.
**Associative model**

The associative model consists of two structures, items and links, both of which have unique identifiers. Items are atomic while links consist of three identifiers to items, links or a mixture of items and links.

Disadvantages:
- Relatively new approach.
- Some sources suggest queries requires more data reads than relational model.

Advantages:
- Semi-structured.
- Item values are not limited by predefined schemes.
- Entities can have several attributes.

**Graph / Network model**

Graph models employ graph theory to structure data. Entities and attributes are represented as nodes while edges represent relationships between the nodes. Hierarchical and network models are subsets of graph models and were popular in the 1980s. The capability of navigating the graphs can be used to increase efficiency of queries compared with the declarative approach employed by relational models.

Disadvantages:
- Less efficient than relational model when one operation is performed on many elements.
- Does not possess the economical backing of relational model.

Advantages:
- Semi-structured, flexible attribute types and entities.
- Scalable
- Navigational interface allows for efficient queries.
- Inherently efficient with graph-like queries.

**Object-oriented model**

In object-oriented models entities are objects, as in object-oriented programming these can have members and methods associated with them. The model allows for navigational processing of data by following object pointers as with the graph model. Objects can also change over time by adding new members or methods and thus allows for an evolving structure.

Disadvantages:
- Still in the process of being adopted by organizations.
- Commercial implementations not widely used as the relational model.
Advantages:

- Semi-structured, objects can be programmed to behave differently.
- Navigational interface allows for efficient operations.
- Can be made highly efficient with queries requesting lots of data on one item.

**Document model**

In the document model each entity is a document containing key/value pairs and as such this model is reminiscent of a file system. Implementations often index the documents in a table to speed up querying of the documents.

Disadvantages:

- Less efficient with attribute-centered queries.
- No inherent constraint checking e.g. non-null.

Advantages:

- Efficient with entity-centered queries.
- Semi-structured, not dependent on predefined schemes.

**Conclusion**

Inspecting the above survey it becomes clear that all models, bar the relational model, allow for semi-structured data. This is a key benefit of those models since different metrics have varying structure. In addition, because new metrics may rely on unforeseen data structures the semi-structure would also simplify future updates of the data management. In section 4.4 we return to this subject when discussing the practical implementation of the framework data management.

**2.5 Scalability**

Web applications and in particular Internet applications often rely on centralized systems to serve many concurrent clients. A weakness inherent in central components is that if they are not able to process requests at the rate they are generated they become system bottlenecks. This affects all parts of the system which depend on the component and may, in the worst case, cause the component to stop functioning altogether. Scalability is a broad term for how well a system copes with changes in its workload. To limit the scope of this section only a few select techniques which can improve scalability are discussed.

**2.5.1 Database**

A common component of enterprise applications dependent on large data sets is a database. In this section two sources of detrimental system impacts inherent in databases are explained together with techniques used to negate them.

**IO operations**

Most database instructions such as selects and updates trigger input/output (IO) operations
which are among the slowest in computers today (Shee, Deshpande & Gopalakrishnan 2004). Commercial databases generally rely on one the slowest forms of computer memory to store data, the hard disk drive. Even with the departure from mechanical drives with the recent availability of cost efficient solid state drives (SSD), hard drives still lag behind other types of computer memory. In systems where the database is a central part this can have severe effects on the overall system performance. Thus, alleviating this limitation is a pressing matter for many organizations. With the steady increase in the size of Random Access Memory (RAM) it is common for databases to hold parts or the whole data set in the faster memory. RAM is however volatile, meaning data is lost when power is turned off or the system is rebooted and thus requires other devices for long term storage. Nonetheless, loading data sets into RAM is becoming increasingly popular and there exist memory caching programming libraries with support for numerous databases.

Sequential operations
Software systems relying on data sets generally require the data to be stored in a reliable and consistent fashion. The concept of Atomicity Consistency Isolation Durability (ACID) has evolved for this purpose and outlines a set of properties which guarantee reliable data persistence. A consequence of ACID properties is that many database transactions are required to be sequential. This effectively puts a limit on database throughput to how fast a single transaction can be executed. Currently a general solution to this problem applicable to all data models are scarce if at all available. A temporary solution is to split the data set between several database instances and this is referred to as horizontal partitioning or alternatively, sharding. Several database vendors offer this functionality with their products out of the box. More commonly, however, databases are offered without support for sharding, forcing customers to implement their own sharding strategy. This can be achieved by implementing a software layer above the database which has the task of spreading the data between several machines. In either case, the method of splitting up data over more than one machine has its limitations. Splitting up the data between separate physical machines results in poor performance with transactions spanning data located on two or more machines. Further, as the data set grows new machines need to be added and the approach is therefore only a temporary fix.

2.5.2 Server
Another common component of centralized systems are servers and in particular Web servers. Individual servers have a limit on how many requests they can handle concurrently. When the limit is reached further requests are commonly ignored. From a clients perspective this may result in a Web page not fully loading or transactions which have no effect. As with databases a common approach to alleviate server overload is by adding additional physical machines, referred to as horizontal partitioning. This can be achieved by running replicas of the Web server on additional physical machines. By routing requests between the machines the system as a whole can potentially handle more requests. The request routing can be done statically, meaning requests from a certain client or of a specific type are always routed to the same machine. Dynamic routing on the other hand takes into account factors such as current workload of the individual machines when routing requests. Similarly to database sharding this approach is not always trivial to implement but by fulfilling certain conditions the process of introducing additional machines can be made easier. One of these conditions is whether the service hosted on the Web server is stateless or not.
Statelessness

Web services sometimes store information between requests from the same client to enhance functionality and user experience, the technique is called server-side sessions. For example, by storing user settings on the server the layout of a Web site may be customized to each user. However, session states can make horizontal scaling more complicated, let us give an example to illustrate the problem.

Alice visits a Web site and is presented with a language selection screen. She makes her choice and a HTTP request containing her selection is sent, under the hood, to the Web server. The server initiates a server-side session and stores Alice’s language selection as a session state. This states is then available to future requests from Alice and can potentially be used to present the Web site in Alice’s preferred language. Let us assume that this is the case and Alice is hereafter presented with text in her preferred language. After browsing the Web site for a while Alice suddenly is faced with a page on the same site but not in her preferred language. Unbeknownst to her, the last Web page request has been routed to another physical machine than her initial requests. This second machine does not have access to sessions stored on the initial server and falls back on the default language when processing Alice’s request.

The above example illustrates a simplified picture of a stateful Web service and one of the issues, if only minor, it gives rise to. Note that it is not the presence of a state giving rise to the issue but rather the fact that the state is stored on the server. With a small change to how the language selection is saved it is possible to avoid the issue and similar ones associated with stateful Web services. For instance, by appending the language selection to each of Alice’s HTTP requests in the form of query parameters which make up the trailing end of a URL. This removes the need to store it on the server and no matter which physical machine processes her request that server will know which language Alice prefers.

In short, by keeping Web services stateless future expansions of the service with additional servers is made substantially less complicated. Statelessness is not always possible but it is a property which should be sought after and departure from it requires justification.
3 Implementation

As part of this thesis the author implemented the framework and monitoring tool as they are specified in section 1.4. This chapter presents implementation details of those systems to explain the internal operations of the system.

3.1 System model

In this section we present a birds eye view of the implemented system which encompasses the monitoring tool and the surrounding framework.

Figure 3. Framework deployment diagram.

Figure 3 depicts a deployment diagram of the prototype framework and monitoring tool. The
diagram shows the hardware devices used in the system, what software artifacts they contain and how they interconnect with one another. The set up is common for Web applications relying on central data sets where a Web server acts as a controller directing information flow between clients. The strength of this structure is its ability to add additional modules or exchange existing ones without affecting other the end points of the system. For instance, a change of database only requires changes to be made to internal aspects of the database interface. The monitoring tool and presentation nodes, however, remain unaffected as long as external aspects of the database interface are unchanged.

Any increase in the workload within the model is expected to originate from increased number of requests from the monitoring tool and presentation node. To manage this the model allows for scaling by installing so called load balancers on the HTTP connection paths. Load balancers have the task of intercepting incoming requests and divide them among several Web servers. In the following sections further details are presented on each of the five entities of the model.

3.2 Monitoring tool

Arguably the crown piece of the thesis project and the primary contribution of this paper is the monitoring tool implementation. The tool employs the page tagging method described in section 2.1.1 to monitor run-time execution of TV apps. There are many similarities with Web pages that TV apps exhibit and in some cases the apps are nothing but a Web page. In addition, Web analytics has had ample time to explore alternative approaches and still page tagging remains the most popular Web page monitoring approach. These two factors have been the main reason for employing page tagging during the thesis project. Note that the scope of the thesis project limits tool implementations to devices from the manufacturers specified in constraint R7. In practice, supporting three different manufacturers was solved by implementing three versions of the monitoring tool. The following sections point out when the three implementations differ in a fundamental way. Apart from this, the descriptions strive to be independent of the different implementations. To support the understanding of this section a partial communication model of the framework which the tool relies on is presented in figure 4.

![Figure 4. Monitoring communication model.](image)
3.2.1 Deployment

To summarize page tagging a sequential list of how it is implemented within the monitoring tool framework follows:

1. A HTML script tag is inserted into the HTML documents of a TV app. The source property of the tag is a URL to a JavaScript file hosted on the framework Web server.
2. When the HTML document is accessed on a Connected TV the above script tag generates a HTTP GET request for the source file.
3. Upon receiving the script file it is executed in parallel with the TV app.
4. The monitoring tool is thus deployed and running within the TV app context.

Let us look at each of the above steps in more detail to get a firmer grasp of what they imply.

Step one

The first step is fairly straightforward from the app developers viewpoint, all they need to do is to insert a predefined line of code into the app’s HTML documents. The exact procedure is described in a text file containing step by step instructions, for curious readers the file is included as appendix A. Performing the procedure takes a few minutes for an inexperienced users and less than a minute for accustomed users. Below is an example of a HTML script tag:

```html
<script type="text/javascript" src="http://domain.com/agent.js?appid=foo"></script>
```

The interesting part is found next to the `src` property which declares where the file containing the script is located, any legal URL is accepted. Notice the query string which begins after the question mark in the URL, it contains a set of key/value pairs, one pair in this case. The query string is sent along with the HTTP request but is not part of the URL per se. The string can be viewed as a set of function parameters common in procedural programming. This specific pair defines an identifier which is unique to each TV app and has to be manually replaced with an appropriate value by the developer. The identifier is later used to trace which TV app submitted a particular data entry to the Web server.

Although this example presumes the app employs HTML this is not a requisite and in practice this is decided by the API of the hosting Connected TV device. From the three manufacturers the prototype tool is implemented for, LG and Samsung support HTML documents. Panasonic on the other hand relies on a proprietary API, implementing similar concepts to those provided by HTML, including a procedure for loading remote script files. The Panasonic implementation thus differs from the other two and this is reflected in the integration instructions. Further, in this example the monitoring tool is implemented in JavaScript. Again this is not a requisite of the framework but purely dependent on TV manufacturer specifications.

Step two

Continuing with the second step, similar to browsing the Web using a computer, users can browse TV apps on their Connected TV. In the same manner Web pages consist of different resources other than plain text such as graphics and video, TV apps also contain various resources. These resources are not included in full within the HTML document but are only referenced with a location defined by a URL. When the browser encounters such a reference it
sends a request for the resource to the domain of the URL. This is done for all referenced content including scripts and this allows the monitoring tool to be loaded with a single line of code. The document server hosted on the framework Web server is responsible for providing the correct implementation of the monitoring tool. This allows TV apps to rely on the same URL when requesting the tool independent of which TV brand they run on. The server procedure is explained in detail in section 3.5.1.

**Step three**

Moving on to the next step, in one regard it is possible to view script tags as programming libraries commonly found in languages such as Java. Unlike libraries, contents of a script file are executed merely by the inclusion of the script, in essence invoking themselves. The tool takes advantage of this fact and registers a handler with the onload event of the HTML document which is triggered when all resources have been loaded. Registering functions with this particular event is common therfore the monitoring tool must handle this gracefully when DOM level 0 events are used. A simple trick to register several handlers to the same DOM level 0 event is to wrap handlers within a wrapper handler which is dynamically created. The monitoring tool uses this approach to avoid trashing handlers of DOM level 0 events already registered by other scripts. The procedure, which is called wiring handlers, is explored in more detail in section 3.2.2.

**Step four**

The last step is maintained until the user either exits the TV app or another HTML document is loaded at which point context is changed. In the first cases there is no reason for the monitoring tool to continue executing but in the latter it ought to, at least in theory, continue. A solution to this is to include the monitoring script in all HTML documents the TV app consist of thus covering all contexts. Considering how TV apps are commonly implemented they more often than not consist of a single HTML document which is acted upon by various scripts. This makes the process of including the monitoring script in several HTML documents only a minor inconvenience for a few special cases.

### 3.2.2 Monitoring

The main task of the monitoring tool is to track various software metrics associated with the execution of a TV app. Up until now, only aspects concerning the deployment of the tool have been discussed, in this section the focus is shifted to the monitoring task instead. The software metrics monitored by the tool implementations can be split up into two types, static and dynamic. Static metrics do not change over the session cycle of a TV app, these are aspects such as device IP address and device firmware version. It is therefore sufficient to check static metrics only once during an app session. Static metrics are an efficient source of information as they require limited amount of system resources to track. Dynamic metrics on the other hand either change or are generated during app execution. These are aspects such as run-time errors and video stream properties. Dynamic metrics require tracking throughout the execution of a TV app to achieve a complete picture of the app execution.

**Static metrics**

The procedure used by the tool implementation to monitor static metrics is to query relevant
properties of the TV API during the loading phase of the app. This consists of an ordinary look up of property values. Which properties are available for querying is dependent on TV API and is one reason for differing tool implementations. For instance, the resolution of the TV app is stored as two numbers in LG and Samsung APIs. These can be referenced with dot notation in the following manner; document.screen.width, document.screen.height. Some properties cannot be referenced directly but require function calls to the API which return property values but the principle is the same. The tool implementations query static metrics as the final step of initialization and is performed by the handler registered to the onload event.

**Dynamic metrics**

The monitoring tool relies on event triggers to monitor dynamic metrics. This is achieved by registering event handlers for relevant events during tool initialization. These handlers are then able to track associated events when they occur during execution. For instance, by registering a handler with media player error events it is possible to report details about those errors if they occur. Keeping track of the amount and type of errors occurring constitutes a good metric to observe and is an example of a dynamic metric. As pointed out in Step three of section 3.2.1, when registering event handlers special care must be taken when dealing with DOM level 0 events. The problem is that many TV APIs do not support level 2 events and therefore handling level 0 events gracefully becomes unavoidable. As previously suggested one solution to do this is to wrap event handlers in dynamically defined handlers. This guarantees that all event handler registered by the TV app are called when associated event is triggered, but there is a flaw with this approach. To see this, imagine an API only supporting level 0 events and the monitoring tool finds that a particular event already has a registered event handler. It wishes to register its own handler with this event and therefore wraps the two handlers in a new handler, so far so good. What if the TV app at a later point in execution registers a new event handler with that same event? Then the dynamically defined handler would be discarded and the monitoring handler would from then on never be called when the event was triggered.

The above scenario is not unheard of and restricting developers to only register event handlers prior to the monitoring script initialization is unacceptable. In order to avoid this issue the monitoring tool arranges something akin to a parallel thread by using the setInterval command available in JavaScript. The command accepts a function name and a value in milliseconds as its parameters and schedules the function to execute at the submitted interval. Algorithm A presents pseudo-code for the function which the monitoring tool submits to the setInterval command during its initialization. The function allows the monitoring script to remedy the issue with lost monitoring handlers when a TV app unwittingly overwrites them.

**Algorithm A. Pseudo-code for ensuring consistency of registered event handlers.**

```
for each event in registered
    if document[event.label] != event.handler
        var temp = document[event.label]
        document[event.label] = function(){ temp(); event.tracker(); };
        event.handler = document[event.label];
    else
        nop
```

The gist of the pseudo-code is that all events which the monitoring tool is tracking are
checked for consistency. If the check fails a new handler is defined which wraps the currently registered handler with the monitoring handler. The algorithm relies on two externals, \texttt{registered} which is list of \texttt{event} objects, the list is created during tool initialization and is device dependent. In the prototype implementation the interval is set to once every second which had negligible performance impact.

3.2.3 Reporting

The final task performed by the monitoring script is to transmit data on the metrics it is tracking in order to persist it in a data set. The framework database is accessed through the interface located on the framework Web server depicted in figure 3. The tool implementations use HTTP requests to transmit data but in order to do so it has to circumvent a limitation imposed on JavaScript.

Cross-domain requests

JavaScript can communicate with remote servers using HTTP requests by using an object commonly referred to as XMLHttpRequest. It is implemented in the majority of today’s browsers with slight variations between them. The issue is that this object is restricted to sending requests to the same domain which hosts the HTML document. This does not necessarily coincide with the Web server domain hosting the framework database interface. There are several workarounds which are commonly employed by Web page developers to circumvent this limitation. The specific method employed by the tool implementation is supported by the majority of Web browsers and is often used by commercial Web analytics solutions. It relies on the HTML IMG element which has a source property for the location of the image file which it refers to. This property can be any valid URL and the location does not have to be an image or on the same domain as the HTML document. Further, every time the property is set or updated a new HTTP GET request is sent to the domain of the URL. The setting and updating can be performed with JavaScript in browsers implementing any basic document object model. Thus, by manipulating the source property of an IMG element it is possible to generate arbitrary HTTP GET requests in JavaScript. Merely sending a GET request does not transmit any relevant information to the Web server. To send data with the request specific key/value pairs are attached to the query string of the URL, the structure of the key/value pairs is specified in appendix C. The query string does not affect the destination of a URL but the string can be accessed by server logic deployed at the destination, in this case the database interface. Note that this workaround is only required for TV APIs which rely on the XMLHttpRequest specification to generate HTTP requests. This is the case with LG and Samsung while Panasonic provides its own request object which does not suffer cross-domain restrictions.

3.3 Presentation

The framework implementation contains a front-end for analyzing the data generated by the monitoring tool. The front-end is accessed through a Web browser and uses JavaScript and a commercial graph package to present underlying data as graphs. The purpose of the front-end implementation is to act as a prototype for future presentation implementations and therefore lacks some key functionality. Specifically, it lacks any authentication procedures and does not provide any support for generating TV app identifiers.
The core structure of the layout is declared in a HTML document which is hosted on the document server along with JavaScript files and graph package dependencies. The HTML document is very basic and its implementation is therefore omitted from this paper. The main body of logic required by the front-end resides in the JavaScript files and files required by the graph package. Appendix B contains a screen capture of the front-end which can aid with comprehension of the following two sections.

### 3.3.1 Scripts

The front-end provides tools to analyze five different metrics which are being tracked by the monitoring tool. Each metric is associated with a JavaScript file which contains methods for querying the database interface and visualizing retrieved data. The methods are fired by user input and transmitted using XMLHttpRequest, the response is parsed and presented with the help of the graph package. The main task of the scripts is to construct queries based on user input and convert data into a form accepted by the graph package. Figure 5 depicts an activity diagram of the standard flow of control between the front-end, user and database interface. Note that the front-end in practice is asynchronous, meaning each user may have several ongoing activities.

### 3.3.2 Graph package

To save time during front-end implementation an evaluation version of a commercial graph package was used for visual presentation. The package goes by the name FusionCharts and uses a JavaScript API to generate Flash based charts. Evaluation versions of the package attach a text string to all generated charts but otherwise reflect the full functionality of the product. The package provides several chart types and the front-end uses three of them; line charts, bar charts and pie charts. Each chart accepts data input as XML or JSON, in addition, to customize the presentation each chart has several properties which can be manipulated.

![Figure 5. Front-end activity diagram.](image-url)

A particular feature of the package the front-end employs is the link property of the chart
elements such as dots in a line chart. The link property can either define a URL or a
JavaScript function name which allows users to interact with chart elements directly. The
front-end uses the link property to accommodate a type of visual data drilling. This is
achieved by populating data into additional charts, there are three charts in total which are
ordered in a hierarchic manner. At the top is a line chart which presents a historical view of all
reported data points for the particular metric being viewed. Below the top chart is the main
chart which contains a customized view of the metric over a selected time frame. To the right
of the main chart is a drill down chart which presents data on the currently selected element of
the main chart. For instance, clicking an element of the main chart populates the drill down
chart with data on the constituent parts of the element. Clicking on the top chart on the other
hand populates the main chart with data for a particular date.

### 3.4 Database

The framework implementation stores the data reported by the monitoring tool through the
database interface. As was made apparent in section 2.4 there exists a wide range of potential
data models to employ. The main criteria when choosing database for the framework
implementation was Java Database Connectivity (JDBC) support of the database. This
originated from the fact that the other parts of the back-end were being implemented using
Java. Beyond this, performance and scalability capabilities had some influence on the
selection process and ultimately MySQL 5.5.8 was employed. A more thorough discussion
pertaining to this outcome is presented in section 4.4. MySQL implements the relational
model, supports JDBC and is available for free under the GNU General Public License. The
database consists of the following tables:

- **Devices** – brand, MAC address and public key of each TV device.
- **MetricTypes** – stores metadata on metrics.
- **Reports** – a group of tables for storing data reported by the monitoring tool.

Figure 6 depicts the relations between the above tables, the remainder of this section presents
more detailed descriptions of each table.

![Database table relations](image)

---

**Figure 6. Database table relations.**
Devices table
When a Connected TV runs the monitoring tool for the first time it is registered in the devices table. A row in the table consists of a unique public key, MAC address and brand of the Connected TV. The table serves the purpose of avoiding excess data in the reports tables.

Reports tables
The reports tables store the data the monitoring tool reports to via the database interface. These tables all have similar structure and the reason for having several is a requisite of the relational model which forbids variations between entries in an individual table. Each row in these tables consist of a unique public key, server time stamp, client time stamp, application id and device id. Further, each individual table has one or more columns for the actual value of the entry. These value columns can be a combination of number and text columns and entries are only stored in the table which has suitable value columns. For instance, screen resolution which is defined by two numbers is stored in a table which has two integer columns as its value columns.

Metric types table
Because data structure varies between metrics it must be stored in a suitable reports table. The metric types table stores information on which table each metric is stored in. A row in this table consists of a unique public key, metric name and the name of the table where the metric is persisted.

3.5 Web server
Monitoring tool deployment, data reporting as well as data presentation all rely on remote resources within the monitoring framework. In order to accommodate this, the prototype framework employs a Web server to host resources and services. Figure 7 depicts a component diagram of the Web server, the remote endpoints and database.

The Web server employed by the framework is Apache Tomcat 6.0 but any Web server able to host documents and Java servlets should suffice. The Web server performs two main tasks, acting as a document server and providing an interface to the database for the remote applications. The framework uses so called servlets to implement Web services. A Java servlet is a Java class which can be used to process HTTP requests by deploying it into a Web server. The servlet must extend the HttpServlet class and should override the doGet function in order to process GET requests. The declaration of which requests a particular servlet is delegated is mapped in an XML configuration file which is Web server specific. In the scope of this paper it suffices to understand that it is possible to map an arbitrary URL within the domain of the server to the servlet. For inquiring minds Rajagopalan, Rajamani & Ramesh (2002) presents a more general introduction to Java servlets.

3.5.1 Document server
The deployment of the monitoring tool and front-end requires certain files to be downloaded onto a Connected TV or computer respectively. These files are hosted on the Web server by the so called document server.
Front-end

The front-end file dependencies are the main HTML document, JavaScript files for requesting data from the database and the graph package files. Because the front-end is accessed through a standard Web browser the files are hosted as simple resources on the Web server. Requests for these files are handled by the Tomcat Web server without requiring authentication.

AgentServlet

As was described in section 3.2.1 the framework relies on a monitoring script which is requested when a TV app is loaded onto the TV. These requests can originate from any brand of Connected TV and since each brand has a particular TV API they all require different monitoring scripts. This is handled by a servlet which intercepts all requests for the monitoring script and identifies which brand of Connected TV made the request. After identification a response is generated with a suitable implementation of the monitoring script. The servlet uses the User-Agent header of HTTP requests for the identification. The User-Agent is a standard header and is a text string which manufacturers are free to use as they please. The AgentServlet uses three regular expressions to identify User-Agent headers from LG, Panasonic and Samsung devices. The following is a list of those expressions:

- LG expression: .*LG browser.*
- Panasonic expression: U2FsdGVkX1.*
- Samsung expression: .*Maple.*Navi.*

Information on specific User-Agent headers is deferred to device documentation.
3.5.2 Database interface

To decouple the database server from the rest of the framework, a custom built database interface is located on the framework Web server accessible through HTTP requests. The interface effectively separates the database from the implementation of other parts of the system. The monitoring tool employs the database interface for reporting data on tracked metrics while the presentation front-end requests the accumulated data. The interface is implemented with two servlets, ReportServlet for submissions and DataServlet for requests.

ReportServlet

The monitoring tool sends data to the database interface through HTTP GET requests and uses the query string as a data container. The structure of the container is specified in appendix C. All requests destined for the report catalog on the Web server are being mapped to the ReportServlet. The servlet performs rudimentary validation checks on the query string and then injects a new entry into the database containing the information in the query string. In order to inject the row into the correct database table the servlet keeps a cached version of the MetricTypes table in memory. If it is not able to find a valid hit in its cache it instead queries the MetricTypes table in the database. Similar procedure is used with the Devices table to registering individual Connected TV devices. The ReportServlet requires permissions to read the MetricTypes table and read/write the Reports and Devices tables.

DataServlet

The presentation front-end of the framework relies on the database interface to provide data for its visualization. This is achieved by HTTP requests destined for the data catalog on the Web server which are being mapped to the DataServlet. Data queries are submitted via the query string of GET requests directed to this catalog. The structure of the query strings used by the interface is specified in appendix D and consists of predefined key/value pairs. The prototype framework implements two query types, both of which are counting queries, that is they count entries in the Reports tables. Both query types use the SQL group by statement and they mainly differ by which column they group by. The group by statement is an operation in SQL which basically groups entries within a query by one of their columns. For instance a group by over a date column would result in a list with the number of entries registered each date. Returning to the two query types, one type groups by date column while the other groups by value column. Both query types also accept additional parameters allowing for filtering by metric type, date, and value columns. The query result generated by the database is sent to the front-end as a string in JSON format which is natively supported by JavaScript.
4 System review

In this chapter the implementation described in chapter 3 is discussed with regard to the requirements presented in section 1.4. Beyond this, discussion about the implementation such as design decisions and limitations which are not appropriate for the previous chapter are deferred to this chapter.

4.1 Requirements

In this section the implementation is verified with regard to the requirements in section 1.4.

Requirement R1

This requirement calls for an unobtrusive cross-device software monitoring tool for Connected TV apps. The following list contains the techniques used to validate the constituent parts of the requirement:

- Cross-device – tool deployment tested on two different TV brands.
- Connected TV apps – tool deployment tested with three commercial TV apps.
- Software monitoring – tool functionality tested on three commercial TV apps.

The tool was tested with three different TV apps which were run both in software emulators and also on two actual Connected TVs. The software emulators which are developed by the TV manufacturers are commonly used by developers to test TV apps. The emulators do not attempt to replicate the hardware aspects but merely the software environment of the TVs. Nonetheless, they are good indicators of how a TV app will function on an actual Connected TV.

Results of the validation indicate that the tool implementation can be deployed on three different TV brands; LG, Panasonic and Samsung. Further, the validation also verified that the tool is able to monitor a set of software metrics on three arbitrary TV apps. The set of metrics consisted of app resolution and URL, video stream buffering duration and bit rate, media player errors and user input via remote control. During the validations neither the writer nor two independent TV app developers were able to notice any difference between running the TV app with or without the tool. The review of the integration instructions asserted that the development impact of integrating the tool in a TV app is negligible.
**Requirement R2**

This requirement defines four parts of the framework which allow the monitoring tool to be purposefully integrated with TV apps. The framework implementation described in chapter 3 is the practical implementation of this requirement. The following list maps sections of the implementation to which part of the requirement they fulfill:

- Tool deployment – 3.2.1 Deployment.
- Tool data interface – 3.2.3 Reporting, 3.5.2 ReportServlet and 3.5.1 AgentServlet.
- Information management – 3.4 Database.
- Presentation – 3.3 Presentation, 3.5.2 DataServlet and 3.5.1 .

The above listed systems and their dependencies were validated in parallel with the validation of requirement R1. That is, by integrating the monitoring tool into a commercial TV app and verifying the operation against the implementation models presented in chapter 3.

**Requirement R3**

This requirement prohibits the monitoring tool from relying on additional hardware beyond a properly installed Connected TV to operate. The monitoring tool implementation described in chapter 3 consists of a JavaScript file and only depends on the Connected TV for its operation. Note that the requirement does not apply to the Web server since it is not dependent on the TVs location. The requirement was validated by limiting the validation of the monitoring tool to standard Connected TV sets.

**Requirement R4**

This requirement prohibits the monitoring tool to rely on software modules which are not part of the API provided by the TV manufacturer. The tool implementation in chapter 3 is tailored to the APIs of the three supported TV brands; LG, Panasonic, Samsung. The requirement was validated by limiting the monitoring tool validation to Connected TVs with default installations.

**R5 - R7**

The remaining entries in section 1.4 are merely constraints to limit the scope of the thesis project and are therefore not verified.

### 4.2 Monitoring tool

In this section the writer reflects on various implementation aspects of the monitoring tool described in chapter 3.

#### 4.2.1 Metric list

As pointed out in section 1.1.2 the main task of a software monitoring tool is to observe and measure software metrics. In a perfect solution a monitoring tool allows for monitoring arbitrary metrics. The monitoring tool implementation has the capability to accommodate arbitrary metrics within the context of a TV app. It requires, however, dropping the
unobtrusive property because it would require developers controlling its behavior in order to do so. This would be achieved by inserting code snippets into the TV app which contains calls to methods of the monitoring script. This is a common approach in Web analytics and more recently Connected TV analytics but has been used sparingly in the prototype framework. Because TV apps often have short and limited development cycle it is common practice to forgo any process which is not required by specifications. Thus, even though it is possible to integrate certain commercial Web analytics products on select TV brands it is forgone due to laborious integration processes.

As described in section 3.2.2 the monitoring tool is able to observe and measure software metrics autonomously by relying on the TV API. Although this dependency introduces limitations on which metrics the tool is able to monitor. During the thesis project a list of metrics was put together which contained metrics which can potentially be monitored by only utilizing the TV API. The list is contained in appendix E and is based on an investigation of the documentation from four different TV brands.

4.2.2 Maintainability

The approach employed for deploying the monitoring tool using a Web based JavaScript file has a number of positive properties which were unforeseen. The tool scripts being served by the AgentServlet can be modified at any time without the need to bring down the Web server. Further, modifications to the scripts do not require manual updating of the TV apps since the script is automatically requested when an app is executed. Connected TV caches, however, can have an adverse effect on this because they allow TVs to use old versions until the cache is cleared.

Another positive aspect of the tool deployment procedure is the frameworks capability to extend its support to additional Connected TVs brands. It can be extended with support for any device which supports HTTP requests by simply implementing a monitoring script for the particular device. This does not actually have to be a script file since the AgentServlet is not limited to scripts but any file which is executable on the target device. In order to keep the implementation unobtrusive, however, the device API must support some means of tracking software metrics with the aid of the API alone.

4.2.3 Limitations

This section discusses some aspects of the monitoring tool implementation which impose limits on its functionality.

Event registering

The tool implementation relies on registering event handlers to monitor dynamic software metrics, as was explained in section 3.2.2. There is, however, a crucial aspect of the method used to manage DOM level 0 events which may result in unexpected behavior of the TV app. This is the case when the TV app employs the same method the tool employs to wrap event handlers. An example to clarify the problem follows.

A TV app is being developed for a shared context and to avoid trashing registered event handlers the developers decided to wrap handlers as described in section 3.2.2. If a monitoring tool was integrated with such a TV app they would take turns wrapping the handlers. Each time one of them registered a new wrapped handler the other would believe its handler had
been deleted. Trying to correct this by wrapping a new handler would thus create a chain of wrapped event handlers which would result in an undefined operation. To avoid this scenario app developers should refrain from wiring handlers and simply replace registered handlers. This is a weakness of the employed method and is currently not enforced by the framework or monitoring tool.

**Reporting**

The monitoring tool relies on the HTTP IMG element to report monitored metrics for the purpose of storage in the database. Although the URL destination of the IMG source property may be of any file type, the client expects a picture in response. This effectively limits meaningful communication using this approach to only sending information in one direction. Fortunately, this is suitable for the tool implementation since the method is only used for one way communication. This is, however, a severe limitation of the method and future tool extensions may require another approach for cross-domain requests.

### 4.3 Web server

The central hub of the framework is the Web server and as such it may be a potential bottleneck in the system. This section looks at server scalability and other implementation aspects of the servlets described in section 3.5.

#### 4.3.1 Scalability

Because of the present growth experienced by the Connected TV domain it becomes vital for any application aimed for that platform to cope with increased demand. This is no different for the framework implementation, in particular its Web services. Defining the full scalability spectrum of the framework Web server is well outside the scope of this paper. This section therefore focuses on the capability of extending the Web server capacity with additional physical servers.

The first task when employing additional servers to manage excess HTTP requests is choosing a procedure for dividing the requests among the servers. As suggested in section 2.5.2 this can be done by routing incoming requests in an equal manner and can be achieved with a round-robin scheme. Splitting the requests equally may not always be the best approach but for the ReportServlet this may actually be very efficient. Since the requests to the ReportServlet are all more or less identical there is little risk of an individual server being overloaded while others are not. When requests require varying degrees of processing a single server could end up with the most resource consuming requests and become overloaded using a round-robin scheme. This can potentially happen with the AgentServlet and the DataServlet in particular, these servlets require a more sophisticated scheme for reliable request routing. However, because the servlets are stateless any routing scheme can be applied. Therefore the discussion of routing scheme is deferred to literature on server scalability.

The practical approach of adding servlet instances is a matter of setting up replicas of the Web server and deploying the same source packages and files. This again is by merit of the statelessness the servlets possess, had they been stateful a method of sharing states would have been required.
4.3.2 Limitations

The AgentServlet uses the User-Agent header of HTTP requests to identify the manufacturer of the requesting Connected TV. Because manufactures may change the contents of this header there is no guarantee that the identification procedure remains correct. In the worst case, the servlet may provide a script which results in a run-time exception on the TV. One way to minimize the risk of misidentifying is to use elaborate regular expressions which do not easily conform to anything but a particular User-Agent. This still, however, does not eliminate the core issue which could potentially still occur. Another safeguard which can be more effective is to have the script validate the manufacturer of the device during its initialization. This is common in Web development where scripts probe various properties to identify Web browser environment. If it is known that a particular browser supports a property and the script environment does not define that property. It can be concluded that the environment is not of that particular browser. Employing this safeguard within the monitoring script, it would promptly halt its execution if it did not recognize its environment.

The ReportServlet and DataServlet both employ data caches to minimize the number of queries sent to the database. This can have negative consequences if cached entries are updated on the database. In the current implementation such updates are not propagated to other servlet caches automatically and this will become an issue if there is more than one servlet instance. A solution to this is to restrict servlet transactions with the cached data to only inserts and selects and prohibit updates. Inserts propagate by virtue of the servlet implementations since entries not found in the local cache results in a database query for that entry. With inserts this will always be the case since they do not exist in the local cache prior to being created.

4.4 Database

Because the framework database is implemented using an off the shelf product it is not scrutinized to the same extent as other framework parts. A note on the choice of employing a relational model and not one of the other models presented in section 2.4. During the design phase of the project it was decided that a document model would be suitable for the framework database. The reasoning was that the flexibility it provided in terms of being semi-structured would simplify integration of the database with the Web server. In addition, a particular open source database employing the document model advertised promising scalability capabilities. The assumption about the integration later proved to be correct but certain database queries were found to take longer than expected. The queries which were affected were select queries which counted the number of entries within the query. The time complexity of these queries was linearly dependent on the number of database entries but had a time coefficient which was not acceptable. As a result the presentation front-end would become increasingly slow as the number of database entries grew. It was thus decided at that point to waive the earlier decision and use MySQL which is a established relational database.
5 Conclusions and recommendations

This closing chapter is concerned with the writer’s point of view as to what the investigation, which has been described in previous chapters, contributes with. In addition, some closing words on certain aspects of the implementation which can benefit from further development.

5.1 Contributions

The first and foremost accomplishment of the thesis project is proving that it is possible to monitor software metrics of Connected TV apps without extensive integration. Previous work in this area, as researched by the writer, requires substantial integration procedures in order to monitor dynamic metrics. The monitoring tool developed during the thesis project is able to monitor a relevant set of dynamic metrics as default by relying on TV API specifications. This makes the tool independent of the TV app it is integrated with and instead makes it dependent on the device which the app is executed on. By doing so, development costs of integrating the tool with apps are reduced. It may however be argued that the costs are shifted to tool development. This is true, but even so the number of TV apps is far greater than the number of TV manufacturers. By careful selection of which brands to support it may be possible to monitor a large number of TV apps while only implementing a handful of monitoring scripts.

Secondly, the monitoring tool framework is a good starting point for any project exploring the possibilities of software analytics for mobile or connected devices. It is evident by the number of recent forays into analytics for mobile phone apps and increasingly TV apps that this is still an evolving domain. Which means there may still exist certain aspects which are yet to be fully explored.

5.2 Next steps

The framework implemented during the thesis project is a prototype, but nonetheless it may serve as a base for a future analytics product. To aid with this there are a number of changes which should be implemented to bring the prototype up to commercial standards. We do not hope to cover them all here and instead focuses on a couple of key tasks.

The current front-end is perhaps the aspect which would require the most attention in order to transform it into a commercial level presentation. As is clear from the screen capture in appendix B the current version is severely lacking graphical design. It also relies on an evaluation license for the data visualization which does not permit commercial use of the current implementation.

As is pointed out in section 4.2.3 the method employed to wire event handlers has its limitations, specifically when developers use a similar approach. Because DOM level 0 events
are deemed to be around for the foreseeable future it is prudent to support them. Relying on app developers may not be sufficient, instead a programmatic approach restricting the growth of the wrapper functions should be employed. Alternatively, a wholly new method which does not rely on DOM level 0 events for monitoring dynamic aspects could be developed.

The database interface of the framework is a common component of enterprise systems which often require extensive data management. In response to this, several frameworks exist which manage data aspects and decouple the business model from underlying databases. Since these frameworks have the benefit of lengthy development and a large user community they should replace the current database interface.

TV apps take many forms and this is reflected in the variety of software and business metrics which may be of interest when monitoring them. It is therefore important to prioritize development of monitoring scripts which are able to monitor additional metrics beyond the current ones. TV manufacturers may be able to assist with this by extending their TV APIs, beginning with support for all metrics listed in appendix E. Another aspect which may prove vital is the monitoring tools support for additional devices. This is achieved by implementing additional monitoring scripts and steps were taken to accommodate this during development of the framework. Nonetheless, developing monitoring scripts requires intricate knowledge of the device API which it utilizes to monitor apps. This is something app developers are able to contribute with since they have a close relationship with device APIs.

During the thesis project potential future users of the framework had a common question concerning to it. They were curious about the capability of integrating the framework with their present business intelligence suites and in particular Web analytics suites. Thus, a brief study was made to investigate the possibility of integrating the framework with two popular Web analytics products, Google Analytics and Omniture. The result of this sidetrack was that it was indeed possible to integrate the framework with both Omniture and Google Analytics. Omniture provides a procedure for inserting data into its data set and by funneling data collected by the thesis framework to the Omniture data set. It is possible to view metrics gathered by the monitoring tool within the Omniture front-end together with data from other sources. Google Analytics unfortunately does not provide any procedure for inserting data into its data set but instead provides data streams of its data set. Swapping the roles of the Omniture approach would accomplish a similar result with Google Analytics. By subscribing to Google’s data stream it is possible to fuse it with the data collected by the thesis framework in one common front-end.
Bibliography


A Integration manual

This file briefly describes which steps are needed to activate collection of statistical data on a Connected TV app. It is only necessary to read the section of the apps target device.

LG & Samsung

1. Paste the following script tag into the html file of the page that is to be tracked, either in the head or the body as the last element:

   `<script type="text/javascript"

2. Replace XXXXXX with the name of the application in question. If you wish to track media player and other device specific objects make sure those are defined in the same html file as the above script tag.

Panasonic

1. Paste the following two lines into the file containing your main stage, preferably close to the top of the document:

   ```
   add_package_load_path ("metric_agent",
   require("metric_agent");
   ```

2. Replace XXXXXX with the name of the application in question. If you wish to track media player and other device specific objects make sure those are defined in the same file.

3. Paste the following line into the same file as above but below the definition of the stages:

   ```
   Agent.metric_init(this);
   ```
The presentation front-end in action.

User keystreams

155 reports

Total keystream reports

FusionCharts v3.2 Evaluation

% of total clicks

Click depth, limit = 10

FusionCharts v3.2 Evaluation

Key presses at depth 1

OK, 15

LEFT, 5

RIGHT, 5

UP, 4

OTHER, 7
### C Reporting interface

This document defines the interface which is used by the monitoring tool to submit data to the database interface. The tools transmits the data through HTTP GET requests and the data is contained within the URL query string. Each request contains a set of key/value pairs and the below table specifies these pairs for each metric type. Note that some key/values are generic and are attached to all requests, these are grouped under Generic in the table below. The values parameter of the query string is a comma separated string of values (CSV) other values are atomic.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generic</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>appid</td>
<td></td>
<td>Application ID, groups all reports made from the same TV-app, value: unique per app string.</td>
</tr>
<tr>
<td>device</td>
<td></td>
<td>Device fingerprint, identifies from which unique device the report originated, value: unique per device string.</td>
</tr>
<tr>
<td>localtime</td>
<td></td>
<td>Local time of report, value: ms since 1st Jan 1970.</td>
</tr>
<tr>
<td><strong>Screen resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘resolution’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>Horizontal and vertical resolution, the first CSV is the width and the second is the height. Data types: int, int</td>
</tr>
<tr>
<td><strong>Keystream</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘keystream’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>The first CSV is the key code of the pressed key. The second CSV specifies at what position in the key sequence the reported key was pressed, first key pressed is 1, second is 2 and so on. Data types: int, int</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘url’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>URL of the app which is being reported. Data types: string</td>
</tr>
<tr>
<td><strong>Referrer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘referrer’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>URL of the page which referred the user to the TV-app. Data types: string</td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘event’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>Descriptor of the event which occurred. Data types: string</td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘error’</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td>Descriptor of the error which occurred. Data types: string</td>
</tr>
<tr>
<td><strong>Media</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric</td>
<td></td>
<td>Metric type, defines metric type of request, value: ‘media’</td>
</tr>
<tr>
<td>values</td>
<td>Name of the media file which is being played. Data types: string</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Bitrate metric</td>
<td>Metric type, defines metric type of request, value: ‘bitrate’</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td>The first CSV is the name of the media file which is being played. The second CSV is the current bit rate of the media file being played. Data types: string, int</td>
<td></td>
</tr>
<tr>
<td>Buffering metric</td>
<td>Metric type, defines metric type of request, value: ‘buffering’</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td>The length of buffering duration. Data type: int</td>
<td></td>
</tr>
<tr>
<td>Device metric</td>
<td>Metric type, defines metric type of request, value: ‘device’</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td>Specifies which platform the device implements, normally name of the manufacturer. Data types: string</td>
<td></td>
</tr>
</tbody>
</table>
This document defines the interface which is used by the presentation front-end when sending queries to the database interface. The queries are contained in the query string of a HTTP GET request which consists of a set of key/value pairs. The below table defines the structure of the query strings.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appid</td>
<td>Application ID, groups all reports made from the same TV-app, value: string</td>
</tr>
<tr>
<td>metric</td>
<td>Metric type, defines metric type of request, value: string</td>
</tr>
</tbody>
</table>
| query  | Defines type of query, value: **countbyvalues, countbydate**  
**countbyvalues**: uses SQL group by statement on value columns  
**countbydate**: uses SQL group by statement on time stamp of server column |
| startDate | Query is limited to entries reported on or after this date, value: yyyy-MM-dd                                                        |
| endDate | Query is limited to entries reported on or before this date, value: yyyy-MM-dd                                                           |
| value1  | Query is limited to entries where column value1 matches the value of this key, value: metric specific                                           |
| value2  | Query is limited to entries where column value2 matches the value of this key, value: metric specific                                           |
| valueN  | Query is limited to entries where column valueN (N is a number) matches the value of this key, value: metric specific                           |
E Metric list

List of metrics supported by TV APIs from three different manufacturers, filled dots indicate capability to provide measurements on the particular metric.

<table>
<thead>
<tr>
<th>Video stream properties</th>
<th>Brand 1</th>
<th>Brand 2</th>
<th>Brand 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Buffer</strong>; current size of the buffer of streaming media content.</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td><strong>Bit rate</strong>; bit rate of streaming media content.</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Codec</strong>; codec used for media playback.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td><strong>Buffer max size</strong>; the maximum memory amount allowed for buffering of streaming media content.</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td><strong>Initial buffer</strong>; size of buffer when streaming media is allowed to begin playback.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td><strong>Video resolution</strong>; vertical and horizontal resolution of currently playing video content.</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td><strong>Playback mode</strong>; playback mode of video content, full screen, PIP and so forth.</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td><strong>Media player fingerprint</strong>; version of media player used for playback.</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Media content identifier</strong>; name/location of currently playing media file.</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Content server identifier</strong>; host of currently playing streaming content.</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Media content length</strong>; length of currently playing media.</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Playback errors</strong>; error events from media playback device.</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Continued on next page.
<table>
<thead>
<tr>
<th>Device properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device fingerprint</strong>; model</td>
<td>●</td>
</tr>
<tr>
<td>number of TV device.</td>
<td></td>
</tr>
<tr>
<td><strong>Firmware version</strong>; version</td>
<td>○</td>
</tr>
<tr>
<td>number of TV firmware</td>
<td></td>
</tr>
<tr>
<td><strong>TV screen resolution</strong>;</td>
<td>○</td>
</tr>
<tr>
<td>vertical and horizontal</td>
<td></td>
</tr>
<tr>
<td>resolution of TV screen.</td>
<td>●</td>
</tr>
<tr>
<td><strong>3D capability</strong>; TV capability</td>
<td>○</td>
</tr>
<tr>
<td>to playback media containing 3D</td>
<td>●</td>
</tr>
<tr>
<td>extension.</td>
<td></td>
</tr>
<tr>
<td><strong>TV volume</strong>; measure of the</td>
<td>○</td>
</tr>
<tr>
<td>currently set volume for the</td>
<td></td>
</tr>
<tr>
<td>TV speakers.</td>
<td>●</td>
</tr>
<tr>
<td><strong>TV muted</strong>; TV speakers are</td>
<td>○</td>
</tr>
<tr>
<td>muted.</td>
<td></td>
</tr>
<tr>
<td><strong>MAC address</strong>; MAC address of</td>
<td>○</td>
</tr>
<tr>
<td>the TV device.</td>
<td>●</td>
</tr>
<tr>
<td><strong>IP address</strong>; IP address of</td>
<td>○</td>
</tr>
<tr>
<td>the TV device.</td>
<td>●</td>
</tr>
<tr>
<td><strong>Network connection type</strong>;</td>
<td>○</td>
</tr>
<tr>
<td>network connection used to</td>
<td></td>
</tr>
<tr>
<td>connect TV to Internet, Wireless/</td>
<td></td>
</tr>
<tr>
<td>Wired.</td>
<td></td>
</tr>
<tr>
<td><strong>Connection bandwidth</strong>; measure</td>
<td>○ ○ ○</td>
</tr>
<tr>
<td>of how much data can be</td>
<td></td>
</tr>
<tr>
<td>transferred over the customers</td>
<td></td>
</tr>
<tr>
<td>network connection.</td>
<td></td>
</tr>
<tr>
<td><strong>Geographical location</strong>; location of the TV device.</td>
<td>○ ○ ○</td>
</tr>
<tr>
<td><strong>Keyboard input</strong>; TV capability for connecting keyboard input devices.</td>
<td>● ○ ○</td>
</tr>
<tr>
<td><strong>Mouse input</strong>; TV capability for connecting mouse input devices.</td>
<td>○ ○ ●</td>
</tr>
<tr>
<td><strong>Key events</strong>; keys on remote control clicked by user.</td>
<td>● ● ●</td>
</tr>
<tr>
<td><strong>URL</strong>; location of where the app is hosted.</td>
<td>○ ● ●</td>
</tr>
</tbody>
</table>