Dynamic configurable web visualization of complex data relations

Master dissertation

Bart den Haak

Amsterdam, 15 August 2006

One Year Master Course Software Engineering
Dissertation supervisor: dr. J. Vinju
Internship supervisor: G.J. de Vries M.Sc.
Company of Institute: Software Improvement Group

Availability: public domain
Version: 1.0

Hogeschool van Amsterdam   Universiteit van Amsterdam   Vrije Universiteit

Abstract

The Software Improvement Group has developed tools to monitor the technical quality of a software system on the source level, one of these tools is a dashboard application. One of the main features is data source and user interface independence in order to develop a product line. Non-technical users want to easily change and deploy this application, without changing and re-compiling the source code. Currently, this is not possible, because of several limitations. One major limitation is the use of traditional web techniques.
Because of the limitations of traditional web applications, the web user interface can not be function independent from the application logic. Most limitations are related to HTML issues, like missing a dynamic communication mechanism.

Rich Internet application techniques are coming up because they have the same characteristics as traditional desktop clients and provide a dynamic communication mechanism. Rich Internet applications are very useful for building independent web user interfaces and are also able to transform data sets, opening new possibilities within internet application development.
In addition, rich Internet applications seem to be very useful to rapidly create prototypes and speed up the development process to construct run-time configuration wizards for customizing data sets and visualizing relations.

This dissertation proposes a language to configure the variation points defined in the software architecture. With this configuration language, variant user interfaces and data sources can be configured to create a product line dashboard application. According to the configurability definition, provided in this dissertation, the user interface and data source must be replaced without affecting the fundamental structure of the application. By using the proposed configurability measure method the configuration language seems to be successful and can be refined for commercial use.

Acknowledgments

First, I would like to thank my coach, Gerjon de Vries MSc, for his support and assistance throughout this project. I would also like to thank the other members of my committee, dr. Jurgen Vinju and Prof. dr. Paul Klink for their guidance and suggestions.

Backbase is a rich internet language and development of the language and its tools represents the contributions of a number of individuals over several years. The creator of the Backbase language is Gerbert Kaandorp and was developed in June 2000.

Several friends both at University of Amsterdam, CWI and Backbase have helped me during the difficult year of my Master degree. I want to thank my friends, with special thanks to my friends Kiara Coen, Julien Rentrop and Jonathan Witkamp for reviewing.

Finally special thanks to my friends, my family, my parents, sister and brother for their patience and support during the last year that I was away studying.

Table of Contents

1 Introduction 6
 1.1 Terminology 6
 1.2 Background 7
 1.3 Case study 7
 1.4 Variability 8
 1.5 Configurability defined 9
 1.6 Goals and Research questions 10
 1.7 Scope 11
 1.8 Summary of contribution 11
 1.9 Overview of dissertation 12
2 Technical Background 13
 2.1 Problems With Traditional Web Applications 13
 2.2 Client and server paradigms 15
 2.3 Data set visualisation 16
 2.4 Configurable Applications 17
 2.5 Methods to configure 18
 2.6 Abstraction with Declaratives Languages 19
 2.7 Method to measure configurability 19
3 Rich Internet Applications 21
 3.1 Overview of RIA technologies 21
 3.2 Asynchronous JavaScript and XML (AJAX) 22
 3.3 Conclusions 23
4 Approach 25
5 Base architecture 28
 5.1 Design rationale 29
 5.2 Trade-off 29
 5.3 Conclusion 30
6 XML Configuration Language 31
 6.1 Design goals 31
 6.2 XML Language Rationale 31
 6.3 XML Configuration Syntax Explained 31
 6.4 Configuration Language Framework 36
 6.5 Alternatives 39
 6.6 Discussion And Future Work 39
 6.7 Conclusion 39
7 User Interface Prototype 40
 7.1 Selecting the RIA technology 40
 7.2 Development Process – A User Centric Design 40
 7.3 Visualisation of data sets 42
 7.4 Reuse of interface components 45
 7.5 Conclusion 45
8 The Dashboard Pilot 46
 8.1 Conclusion 47
9 Results and Analysis 48
 9.1 Abstract Configuration Language 48
 9.2 Rich Internet applications 48
 9.3 Analysis 49
 9.4 Discussion - Configuration or Programming 50
 9.5 Future work 51
 9.6 Conclusion 51

10 Evaluation 53
10.1 Positive results 53
10.2 Negative results 53
10.3 Results for employer 53
10.4 Reflection on research approach 53
References 54
Appendix A, Pilot Feature Requirements 57
Appendix B, Configuration Language XSD 59

1 Introduction

This chapter describes the background and a case study to provide understanding for why this project was done. Afterwards, the research questions and objectives are defined.

1.1 Terminology

Terminology used in this dissertation is defined below.

Application: The combination of the user interface and the associated underlying logic that implements the functionality visible through the interface [Phanouriou01].

Application Logic: The part of the application but not part of the user interface [Phanouriou01].

A **client** is a computer system that accesses a (remote) service on another computer by some kind of network. The term was first applied to devices that were not capable of running their own stand-alone programs, but could interact with remote computers via a network. These dumb terminals were clients of the time-sharing mainframe computer [Wikipedia06].

CORBA: Common Object Request Broker Architecture. Middleware that defines application programming interfaces, communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA therefore provides platform and location transparency for sharing well-defined objects across a distributed computing platform [Wikipedia06].

Dashboard: A dashboard is a visual display of the most important information needed to achieve one or more objectives, consolidated and arranged on a single screen so the information can be monitored at a glance [Few06].

End user: The person that uses the application's user interface [Phanouriou01].

HTTP: The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.

RIA: Rich Internet Application (see Chapter 3 for more detail).

SOAP: Simple Object Access Protocol is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can build on [Wikipedia06].

Toolkit: A toolkit is the mark-up language or software library upon which an application's user interface runs. Note that the word "toolkit" is used in a more general sense than its traditional use. The term is used to mean both mark-up languages that are capable of representing user interfaces (e.g., WML, XHTML, or VoiceXML) as well as APIs for imperative programming languages (e.g., Java AWT, Java Swing, or C++ MFC) [Phanouriou01].

A **thin client** is a minimal client. Thin clients utilize as few resources on the host computer as possible. A thin client's job is generally just to graphically display information provided by an application server, which performs the bulk of any required data processing [Wikipedia06].

A **thick client** (also known as a FAT client or rich client) is a client that performs the bulk of any data processing operations itself and relies on the server it is associated with primarily for data storage. Although the term usually refers to software, it can also apply to a network computer that has relatively strong processing abilities [Wikipedia06]. Lots of articles and books write the term 'fat' client with capitals as a term joke. This dissertation use the FAT client term for recognition.

Smart clients are "easily deployed and managed client applications that provide an adaptive, responsive and rich interactive experience by leveraging local resources and intelligently connecting to distributed data sources" [MSDN06].

User Interface: The aspects of a computer system or program which can be seen (or heard or otherwise perceived) by the human user, and the commands and mechanisms the user uses to control its operation and input data [DieNet06].

Variability: "Software variability is the ability of a software system or artefact to be changed, customized or configured for use in a particular context" [Bosch04].

W3C: World Wide Web Consortium

XSD: XML Schema Definition

1.2 Background

Analysing source code of large, complex software systems is one of the main activities of the Software Improvement Group (SIG). Based on those analyses, the SIG developed several services and products. The activities of the SIG vary from automatically generated documentation for legacy systems, doing software risk assessments, software renovation management and software monitoring. Thanks to above activities, the SIG has access to source code of several large software systems, written in various programming languages varying from Cobol to C#. To analyse these software systems, the SIG developed several tools. These tools are able to distil important data about a software system, e.g. software metrics data. The SIG itself is an open-source minded organisation, part of their toolset is based on freely available open source techniques (e.g. Eclipse).

1.3 Case study

One of the tools the SIG developed is a web-based software monitor application. This tool acts as a dashboard application to display software metrics data (data relations) stored in a relational database (filled by external tools) to the end user, in the form of charts and tables, specific for a certain system, programming language and customer.
A dashboard application normally shows data from a data source to an end-user by using a (graphical) user interface. Therefore, *data source* and *user interface* are the main features of a dashboard application and the main concerns of the involved stakeholders (see appendix A).

Main concerns
If the metric data must come from another data source (for example XML instead of MySQL) the application logic must be modified by software developers. Also, if another user interface must be used for a different type of customer, the application logic also needs to be changed by software developers. This is a time consuming process and increases development costs and are therefore the main concerns of the involved stakeholders (see appendix A). In order to decrease development cost, the dashboard application logic must be reused, with the ability to connect with different user interfaces and data source variants.

In addition, stakeholders such as user interface developers, system administrators etc., also want to change, customize, install and deploy the application, without changing and recompiling the source code of the application. Thus, a software system which can be configured by several stakeholders. But how can we construct a configurable dashboard application and what is configurability?

1.4 Variability

Variability is related to 'what' and 'when' in the software development life cycle variation will be bound (called binding time), while configurability is 'how' to create variability.
In the case study of this project, the data source and user interface are variable application features and the application logic must be reused (commonality). Reuse of features with the ability to connect with variable features is typical for a software product line [Bass03]. Imagine several variants of the dashboard application in the case study. For example, a combination of user interface A and data source A or user interface B and data source A could be several products in the dashboard product line (see figure 1-1).

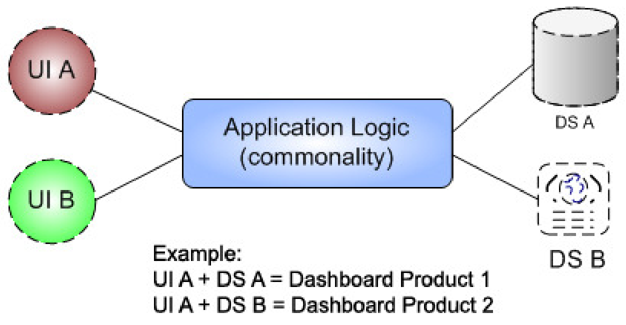

UI A
UI B
Application Logic (commonality)
DS A
DS B
Example:
UI A + DS A = Dashboard Product 1
UI A + DS B = Dashboard Product 2

Figure 1-1, Variability in user interface (UI) and data source (DS)

Variable features require support for variability. When introducing variability into a software product family there are a number of steps to take along the way in order to get the required variability in place, and to take care of it once it is in place [Bass03, Svahnberg01]. This dissertation focuses on the last two steps (see Figure 1-2).

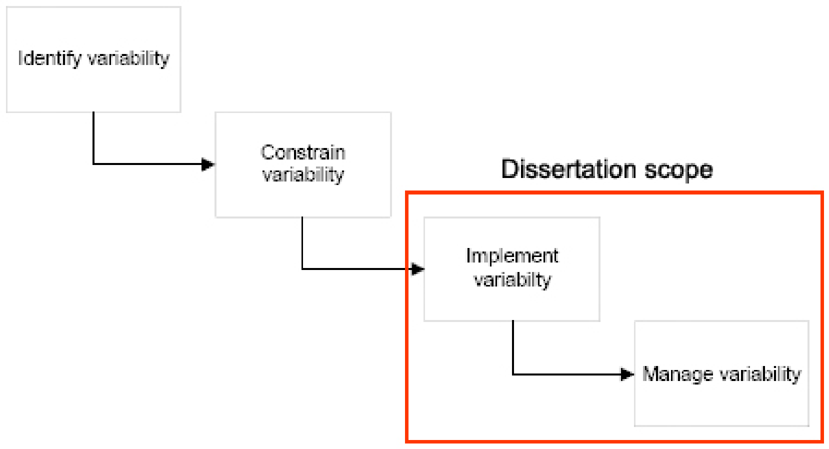


Figure 1-2, Introducing variability

To implement the application variants, modelled by feature diagrams [Svahnberg01], variability realization techniques can be used for artefacts in various stages within the development process, also known as binding time, but are usually applied at pre-deployment time [Bragança04].

Table 1, Software Development Lifecycle activities

Development activities (binding time)	**Artefact**	**Variability Realization Technique**
1. Product architecture derivation	Components Frameworks	Configuration management tools, parameterization, Architecture Reorganization
2. Detailed Design	System Design	Design Patterns, Framing, parameterization
3. Compilation	Source code files	Aggregation, Pre-processor tags, Overloading, parameterization
4. Linking	Build files	Macros and aspects
5. Deployment	*Configuration files*	*Reflection, parameterization*
6. Run-time	*Running application*	*Plug-ins, parameterization*

To get an overview of variation techniques, we combined the development activities (the binding time) with the variation techniques defined in [Anastasopoulos01]. Table 1 shows the different phases of the software development lifecycle (when to bind) and the techniques that are available to vary a certain artefact (what to bind). Also other, more detailed phases, can be defined. This dissertation will concentrate on variability techniques, specific for the dashboard and web-based domain, to support configuration in the *deployment* and *run-time* phases (phase 5 and 6) of the development lifecycle.

1.5 Configurability defined

Configurability is one of many quality attributes (QA) for a software system. There are many configurability definitions available [Losavio03, Firesmith03, Sommerville06].
[Bass03] describes it as *modifiability*, which means: "the ability to make changes to a system quickly and cost effectively", but has much overlap with other configurability definitions. Which one is relevant depends on the stakeholders and

the system properties that are of interest. All definitions have one commonality, which is the ease of configuration, with a minimum of development effort, to change to the original program to fulfill stakeholders' demands.

This dissertation uses a more general definition and defines configurability as follows:

"The system configurability reflects the ability of a system to be adapted to new requirements and operating environments without change to the fundamental structure of the software".

[Ian Sommerville, 2006]

1.5.1 Deployment and run-time configurability

To create affinity with the development lifecycle (or binding times), described in section 1.4, we use the terms "deployment" and "run-time" configurability.

Deployment configurability means the configuration of the application *at deployment time* and **before run-time**, like settings of data source credentials.

Run-time configurability means the configuration of the application *after deployment time* and at **run-time**, like personalisation and customisation.

1.6 Goals and Research questions

Deployment configurability

The source code and fundamental structure of the original dashboard application need to be modified by technical stakeholders (see appendix A) if a different data source must be connected, e.g. XML, MySQL or MS SQL Server for a different customer. The same if a different user interfaces technology, e.g. a Java Swing application instead of a web user interface, must be connected. Normally, only software developers can modify the system, however, other technical stakeholders may want to change and deploy the system (e.g. system administrators) for a different type of customer with different needs.

Run-time configurability

Non-technical stakeholders want to configure the way data relations and data sets are shown in the user interface (customization/personalisation), for example a pie chart instead of a table. Therefore they must configure the application via a user interface to change the different views of data sets and relations. The traditional web dashboard application limits (see Section 2.3) the possibilities to create run-time configurability and cannot function independently from the application logic (chapter 2). Therefore, sharing of the same application logic with other user interfaces is not possible.

The main research question is defined as follows:

How can we design and construct configurable web applications to visualize data relations?

Where configurable means, according to the definition and the dashboard case; the ability to adopt new data source and user interface requirements, without changing the fundamental structure of the application.

In order to answer this main research question, the following sub questions need to be answered:
Q1. Which web technologies contribute to configurable web applications?
Q2. How can we visualize data sets and relations independently from the application logic?
Q3. Which configurability methods are available and are useful?
Q4. How can we measure configurability?
Q5. Which architectures can be used?
Q6. How can we configure variation points to adopt new requirements, without affecting the fundamental structure of the application at deployment time?
Q7. How can we configure data relations and data set visualizations at run-time?

We define the following main objectives:
- Define and select what kind of web technology can support a configurable web application to visualize data relations;
- Define requirements and create a base architecture with variation points, specific for the case;
- Describe the techniques to support the variation points defined in the architecture;
- Describe the proposed method to validate configurability;
- Develop a pilot to validate the first three objectives, based on the validation method proposed;

1.7 Scope

This dissertation primarily focuses on dashboard applications within the web-based domain and configurability implementation techniques for the deployment and run-time phase in the software development lifecycle. Architecture design of product families and feature design techniques are widely discussed in the literature, but not discussed in this dissertation. Also, web techniques used for this project will not be validated, because of the configurability focus.

1.8 Summary of contribution

This dissertation makes the following contributions:
- An overview of available web technologies to support configurability;
- A base architecture, which supports deployment and run-time configurability;
- A dashboard pilot, built out of several prototypes. Every prototype will cover an element of the defined architecture, which supports configurability. It will give insight into the possibilities and functioning of some of the techniques that were discovered during research;
- A specification of the configuration language;
- A method to measure configurability.

From a practical point of view, the following contributions were made for the SIG organisation:
- Requirements of the dashboard web application, specific for the case;
- Description of the construction process for a rich user Internet application by using a user centric design approach.
- A dashboard pilot

1.9 Overview of dissertation

This dissertation proposes a comprehensive solution to the problem of building configurable web applications by using a configuration language, data source and user interface independently. It is organized as follows. Chapter 2 describes the related work upon this dissertation is build. Chapter 3 gives an overview of available rich Internet applications techniques and the advantages of them.

Chapter 4 gives an approach to create configurable web application. It gives the reasoning about the process. Chapter 5 describes the architecture we used to create the configuration language, which respectively is described in chapter 6.
The web-based user interface prototype to support run-time configurability is described in chapter 7. Chapter 8 describes the final web dashboard pilot to validate configurability, based on the method proposed in this dissertation. Chapter 9 presents the results of this project. Finally, chapter 10 evaluates the project.

2 Technical Background

This chapter describes the technical background of this dissertation, which is needed in order to understand the design decisions made in this project. The subjects discussed are the enablers of this research.

2.1 Problems With Traditional Web Applications

Traditional web applications today build upon the page-based model and uses the HyperText Markup Language (HTML) to program the user interface. These applications are also known as "Multi Page Interface" (MPI) applications, because ever interaction with the end user results into a new page (see Figure 2-1).

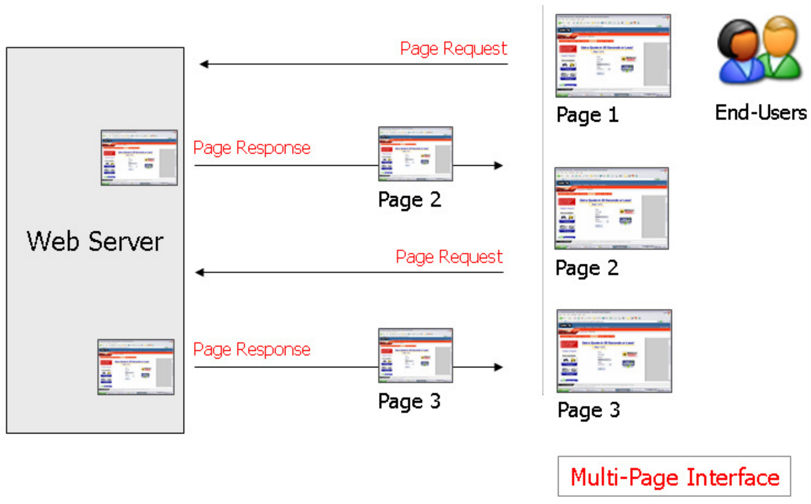


Figure 2-1, Multi Page Interface Model.

Because this model is page-based, there are several main problems [Duhl03, Preciado05] with traditional web applications. In the next sections the most relevant problems are described.

2.1.1 Data complexity

Examining the interrelationship of pieces of data or information illustrates data complexity. For example, departmental and enterprise application users often attempt to make optimal use of internal data, but there is often too much of it. Tools are often limited in their ability to detect patterns in data and make them visible via a Web interface. New ways to interactively visualize data are needed. Interactive data exploration can allow users to navigate through the data and get a better understanding of it. Such tools allow users to look at a chart and then drill down into sections by clicking on them to see increasing levels of detail. When users are able to visualize and manipulate intricately related data, data complexity is reduced or made understandable and compelling by the simplicity and elegance of the visual presentation [Duhl03].

2.1.2 Configuration complexity

Web sites have been unable to present users with a visual picture of custom-built products or present anything other than text for representing the result of a criteria-based or parametric search [Duhl03]. Therefore, it is difficult to create run-time configurable web applications.

2.1.3 Communication complexity

HTML originally was designed to represent content in a structured way and to be easy modify and maintain. This is because the mark-up language abstracts the complex underlying technique, so that not only software engineers can publish content. HTML documents can be linked together by using hyperlinks (HTTP GET) and can send simple value pairs to the server, by using forms submits (HTTP POST).
HTML cannot function as a standalone client, because it is not designed to send and receive dynamic messages to and from the server (like SOAP). In order to create dynamic pages, traditional web applications use (intelligent) server side techniques (e.g. JSP, PHP, ASP), to generate (dumb) web clients (with HTML and JavaScript), where the user interface logic is mixed into the application logic.

2.1.4 Prototyping

HTML can be used to display data in a structured way. It was designed to rapidly publish content on the Internet. Because we try to turn web-pages in to real applications, feature requirements become more complex. HTML cannot handle these complex features. Creating prototypes with complex user interaction with HTML is a very time-consuming process. Interaction designers use other tools to create prototypes for FAT client applications, rather than using HTML [Cooper03, Snyder03]. If they could use interface components off-the-shelf, to rapidly create web-application prototypes, prototype development time would be decreased.

2.1.5 Testability

Because traditional web-applications are server-dependent, its difficult to (automatically) test the user interface logic independently from the application logic [Memon02], they are mixed together. Think of how you should test a JSP page or Java Servlet [Wutka00] separate from the HTML and JavaScript.

2.1.6 Conclusion

Web applications need to be able to represent complexity, allow users to visualize complex data, configure products with multiple options, search large data sets and allow for interactive exchanges between the user and the data. Traditional web applications cannot represent such complexity because of the limitation of HTML pages. In addition, HTML can not function as an independent client, because of the limited communication mechanisms.
If we want to build a configurable web application, we need a web technology, which eliminates traditional web application problems. With the rise of faster desktops computers, browser clients have more processing power and can handle more complex user interface actions. Rich Internet application software claims to solve the problems traditional web application have by moving server computing tasks to the client, for example table sorting. Because rich Internet applications are critical to construct configurable web-application, they will be introduced in a separated chapter (chapter 3).

2.2 Client and server paradigms

Within traditional client/server applications, the client is mostly a FAT client (a mix of application logic and presentation logic). The client connects to the server when, for example, an event occurs (e.g. after selecting a menu item in the menubar) and requests data. But, the client decides *when* a connection is made with the server and *what* message will be sent. Therefore we call this the 'client in control' paradigm.

With traditional web applications, the client is thin, because HTML/JavaScript only supports a limited amount of presentation logic. The server will render the client by each user (page)request (see section 2.1). This paradigm we call 'server in control'.

With rich Internet application software (see chapter 3) you can create powerful rich web clients which act like traditional desktop FAT clients. Web clients do have some limitations: Where traditional desktop clients use sockets, RIA clients use HTTP, which limits data transfers. In addition, most web client are not powerful enough, for example, to transform complex and large data sets with Extensible Stylesheet Language Transformations (XSLT) [Kay04]. You also don't know how much power a web client has. One user can have a Windows 98 desktop machine with a Pentium2 400 Mhz processor and 128 MB internal memory. Another can have Windows XP and a Pentium 4 3,1 GHz processor with 1GB internal memory.

2.2.1 Client in control (FAT Client)

When the client is in control we speak about a 'FAT' or rich client, because all presentation logic (and some application logic) is programmed into the client source code.
Some relevant advantages of a FAT client are:
- Rich User Experience, by means of a better user interface;
- Higher developer productivity;
- Responsive.

Some relevant disadvantages of a FAT client are:
- Tough to deploy, deployment had to be done at multiple location;
- "DLL Hell";
- Client side data transformations can slow down the application.

2.2.2 Server in control (Thin Client)

When the server is in control we speak about a 'thin' client, because all presentation logic is programmed into the server source code. Therefore the server will generate user interface logic (like JSP pages generates HTML pages).
Some relevant advantages of a thin client are:
- Easy to deploy, single location update;
- Easy change management;
- Fast data transformations.

Some relevant disadvantages of a thin client are:
- Network dependency, FAT clients are usually network based;
- Poor user experience, mainly emit HTML;
- Complex to develop, so mainly result in expensive maintenance cost.

2.2.3 Mixed model (Smart Client)

The mixed model is a combination between client and server logic (also knows as smart client) and effects the performance quality attribute. When the performance of the client decreases, you can move the presentation or application logic to the server and vice versa. This ability to move presentation logic, must be provided by configuration to increase maintainability. For example, whether you want data set transformation on the client or at the server. In the case of the dashboard pilot, only the data sets transformations (see Section 2.5) must be configured. With this combination you will able to refine the application to the stakeholders' needs. It is a trade-off between more logic on the client (see Section 8.2.2) and more logic on the server. Figure 2-2 shows the benefits of a smart client and the overlap with FAT and thin clients.

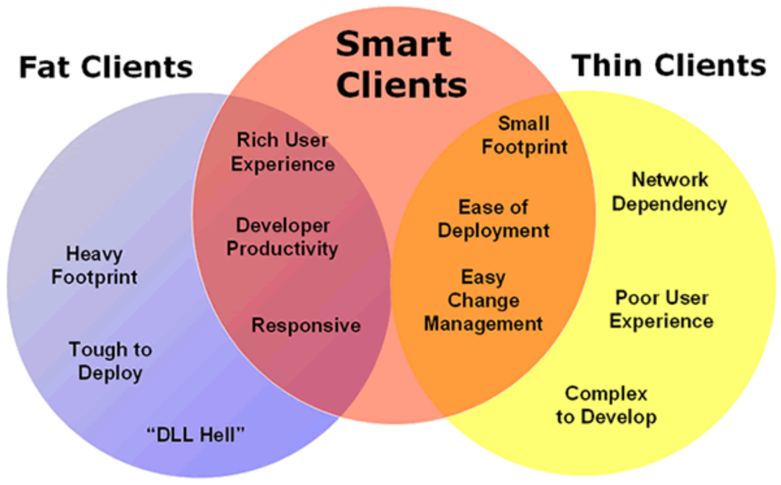


Figure 2-2, Smart clients, a mix of both thin and FAT (adopted from MSDN)

2.3 Data set visualisation

A dashboard application uses charts and tables to represent information from a data source to the end user. Dashboard applications need to fulfill several requirements [Auer03] in order to provide the right information for a individual person in a efficient way. Tables and charts have proved to be successful at representing data sets [Few06]. Which table or chart is most useful depends on the type of dashboard and end-user's demands. In order to fulfills these demands, it is useful for every user to customize their data set visualisation (run-time configurability).
Development of chart and tables to visualize data sets is a very time-consuming process, but there are many off-the-shelf components available on the internet. Selection of those component depends on your requirements (see Section 7.3.2).

Data Transformations
To create data visualisation we need to transform data into a visual representation. XML is a universal format for storing data [Harold04]. Since we want to construct a data source independent application, XML seems the best technique to use due to the wide support of this standard.
Converting XML data into a visual representation can be done by using Extensible Stylesheet Language Transformations (XSLT) [Kay04]. There are many freely available XSLT parsers, mostly integrated into the development libraries (like the .NET framework [MSDN06] or the Java standard framework [Wutka00]). XSLT transformation can be used *client-side* and *server-side* with the exception of thin clients, like HTML or WML clients.

2.4 Configurable Applications

As said before, a dashboard application normally shows data from a data source to an end-user by using a (graphical) user interface. Therefore, **data source** and **user interface** are the main features of a dashboard application and therefore the main concerns of the involved stakeholders (see appendix A).
With the definition of *configurability* in mind, we want to change above features without changing the fundamental structure of the program. In our case, this also means without changing the source code. To support configurable web applications according for our case we need:

- Data source independence, to support multiple variants of data sources.
- User interface independence, to support multiple variants of user interfaces

2.4.1 Data source independency

One of the features defined is data source independence (DSI), therefore we also describe this part. Because this dissertation primarily focuses on configurability and documentation of DSI is widely available, this subject will not described in detail. Only the possibilities of configurability will be highlighted. Architectures to support DSI are not new and therefore many frameworks are available which support data source independence (like J2EE and .NET frameworks).

One major disadvantage of the available frameworks is that you cannot connect to a data source other than standards (XML, databases/spreadsheets, Java objects). When you want to connect to a custom data source (like an ERP framework or in our case, with a custom metrics data framework), things become more complex and difficult.
Data source independent API's like Java Database Connectivity (JDBC) and Lightweight Directory Access Protocol (LDAP) are no longer useful. What if you want to connect for example to Hibernate [Hibernate06], Castor [Castor06] or other object/mapping frameworks?

Microsoft has introduced the adapter model within the .NET framework 2.0 [MSDN06]. Sun has introduced the Data Access Object pattern DOA [Sun06]. If you want to use another data source, you only have to write a custom data adapter, which implements a given interface. This principle will also be used in this project. It does not matter which data source framework you use anymore, you only need to write an adapter for a given data source once.
If you want to use the API's like JDBC, Hibernate or the LDAP library (e.g. to connect with a spreadsheet program or plain XML), you can still use them in your custom adaptor. You only have to implement the interface.
Benefits of data adapters are:

- Greater transparency between business, data access, and data source layers.
- Centralized all data access in a single layer. All database access is in the Adapter.
- Increased maintainability.
- All data queries can be stored externally, like in a configuration source.
- Reduced complexity of business layer code.
- Programmers need to learn only one data source interface.
- Reduced the total amount of code required to access a data source.

One disadvantage is that you have to write a significant amount of repetitive source code to construct a data adapter, but you only have to write it once for every data source. For many developers, this disadvantage is good enough reason to ignore the long-term benefits of using a framework-neutral API, especially where there are strict project deadlines [CodeFutures06]. But products like FireStorm/DAO and Visual Studio.NET provide data adapter code generation which increase development speed.

A data provider will select the right adapter based on the configuration of the application. This is a variability point in the architecture to support data source independence and will also be used in the configuration language (see chapter 5).
The framework prototype in this dissertation uses a custom adapter framework, because of the short project duration. This custom adapter framework could easily replaced by, for example, Sun's J2EE DAO.

2.4.2 User interface independency

To create user interface independency, a dynamic communication mechanism must be used to connect with the underlying application logic, for example, sockets or SOAP. This can be done by using intelligent clients (smart and FAT clients, see Section 2.2.1). Since all rich internet application techniques provide XML communication techniques, a SOAP-like protocol is most likely.
To create user interface indecency for all type of clients (thin, smart and FAT), abstract user interfaces languages and models, for example UIML [Phanouriou01] and XIML [Goschka01], seem to be the solution. Abstract user interface languages are meta languages to describe the user interface at a higher level of abstraction, but these languages are still in their infancy. [Phanouriou01] and [Goschka01] propose user interface source code generation (e.g. HTML or Java Swing) out of the abstract user interface definitions. Then automatic generated proxies will connect the generated user interfaces with the underlying application logic. Abstract user interface languages only support user interface independence (only one variation point) and cannot be combined with other configuration techniques in order to support multiple variation points (like data source and user interface independence).

2.5 Methods to configure

As said before, configuration of the dashboard application of the case study will take place in the deployment and run-time phases of the software development lifecycle.

Deployment configuration
Web server vendors like Microsoft's Internet Information Service (IIS) and Apache (with tomcat Java container) uses XML configuration files, for example *web.config* in IIS and *config.properties* in Apache TomCat, to configure environments settings (also known as parameterization, see table 1-1).
Traditional client/server applications often use initialization files (INI files) to configure limited application deployment options. But more application vendors (e.g. Oracle, SAP, SQL Server), and framework vendors, like the MS.NET framework, seem to use XML as configuration language more widely.

Run-time configuration
Run-time configuration is mostly done via the user interface, since the application is in run-time mode. Run-time configuration affects multiple sub-quality factors [Firesmith03], for example internationalization (also known as globalization and localization) and personalization. With the abilities of FAT clients, more complex user interaction can be used to support complex run-time configuration (like plug-in installation). Most run-time configuration in FAT clients is done by using configuration dialog screens (think of the settings screen of MS Word) or via (intelligent) wizards (think of your e-mail account configuration or plug-and-play

wizard), also known as user interface agents [Dryer97]. Therefore, these proven techniques will also be used in the application build for this project.

2.6 Abstraction with Declaratives Languages

The Internet (developed in the ’50s) was original designed to exchange information. Because only technical engineers could program the information ‘pages’ they invented HTML, the first declarative web user interface language. The idea, the abstraction of complex technology with HTML, was to give journalists the possibility to publish information onto the revolutionary Internet.
They added scripting and plug-in functionalities (’80s) to the clients (browsers). We are at a point where these old techniques have reached their limitations and become too complex. This abstraction paradigm will continue and this is why another level of abstraction is coming up (figure 2.3). The languages of rich Internet applications. For example, this dissertation uses the Backbase declarative language, called BXML. Now even techniques are available to abstract all user interface languages into one *abstract user interface language* (also mentioned in chapter 2.4.2), most are also declarative. However, why are these languages declarative and based on XML?

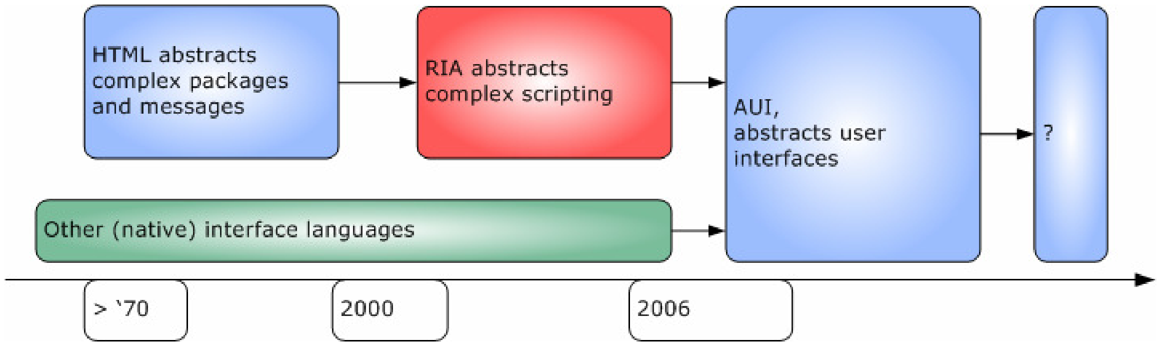

HTML abstracts complex packages and messages
RIA abstracts complex scripting
AUI, abstracts user interfaces
?
Other (native) interface languages
> ‘70
2000
2006

Figure 2-3, Moving abstraction levels

Many of the new declarative languages obtain their syntax from the eXtensible Markup Language [Harold04]. XML facilitates the creation of new vocabularies that describe domain-specific content and context, organized into hierarchical information structures. XML has become the official meta-language for information on the Internet. It is a meta-language because it can be used to define other languages that are relevant to various application domains by providing a common syntax. Therefore, many domain specific languages (DSL) are using an XML based grammar.
Although declarative languages are not yet ready for mission-critical applications or high-performance computing [Phanouriou01], this dissertation shows that they are ready for building user interfaces and can provide a useful level of abstraction to create a configurable language.

2.7 Method to measure configurability

At the moment, we could not find any methods in the literature about measuring configurability. Because configurability has much overlap with modifiability (see section 1.5), we researched on how modifiability is measured. It is measured by using specific changes as benchmarks and recording how expensive those changes are to make [Bass03]. Based on this method and the definition of configurability (see section 1.5) we propose the following method to measure configurability of the dashboard application (see Figure 2.4 for a schematic overview):

1. Define the fundamental structure of the application and select the source files belonging to this structure. The fundamental structure are the components within the application architecture which are not identified as variable;
2. Replace the architectural components in which variability points are defined. For example, if the user interface component is defined as a variation point, replace the web user interface with a desktop user interface.
3. Count the files in the fundamental structure which are changed. For example, by using the date/time properties of the files before and after the change.
4. If the total count of changed files in the fundamental structure equals 0, then we assume the fundament structure has not been changed and the application is configurable.

The configurability definition defined in section 1.5 only affects the deployment configurability, because measuring run-time configurability makes no sense, since the structure of the application will never change at run-time.

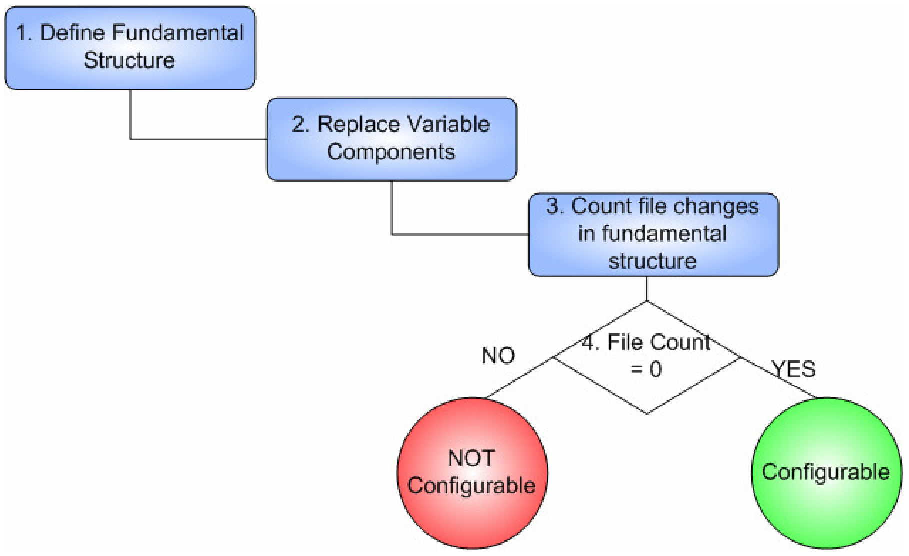

1. Define Fundamental Structure
2. Replace Variable Components
3. Count file changes in fundamental structure
4. File Count = 0
NO
YES
NOT Configurable
Configurable

Figure 2-4, Schematic view of the method to measure configurability

3 Rich Internet Applications

Traditional web applications limits (see chapter 2.1) the possibilities to create run-time configurability and cannot function independently from the application logic. Therefore, research on rich Internet application technologies was done. The relevant results are described in the following sections.

Rich Internet Applications (RIA) are web applications that have the features and functionality of traditional desktop applications. RIA's typically transfer the processing necessary for the user interface to the web client but keep the bulk of the data (i.e. maintaining the state of the program, the data etc) back on the application server. Because of this model, the web browser does not need to refresh every page request, only the relevant user interface components (see figure 3.1)] will be updated. In contrast to traditional web applications, rich Internet applications are based on the single page (SPI) model.

Because the user experience becomes increasingly important, we see a big explosion of rich Internet applications. Desktop applications are moving towards the Internet because of the new "rich" possibilities of current web browsers. The web browsers of today act like a new mini platform (some call it Web 2.0) on which we can build "rich clients".

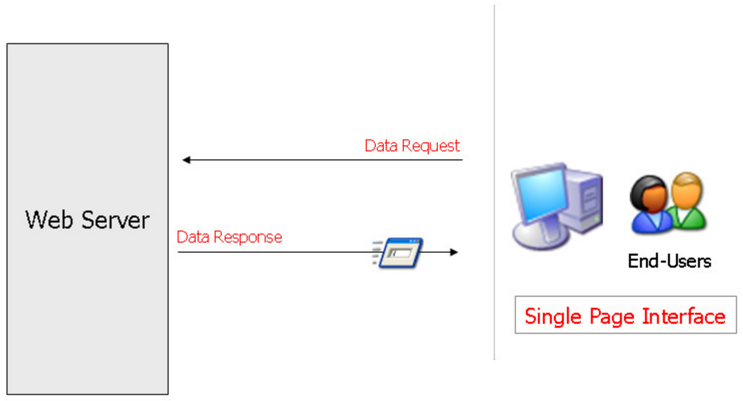


Figure 3-1, Single Page Interface Model

3.1 Overview of RIA technologies

Because RIA is very young, there are not many techniques available at present. Table 2 gives an overview of available RIA technologies, browser compatibility, programming language and if the technology can support the reuse of user interface components. In addition, the communication technique with the server is provided (see Section 3.3.3). Reuse of user interface components is important to speed up the development process after prototype development. If user interface components can be created in prototype development, they can be reused in the final application, which increases development speed.

Table 2, Available RIA technologies

Technology name	**Browser need plug-in?**	**Language**	**Reuse of components**	**Communication technique**
Java Applets	Yes	Java	Yes	TCP Sockets
Adobe/Macromedia Flex[Macromedia06] Laszlo [Laszlo06]	Yes	MXML	Yes	TCP Sockets
Mozilla XUL [XUL06]	Yes	XUL	Yes	TCP Sockets
Microsoft XAML (not available yet) [MSDN06]	Yes	XAML	Yes	TCP Sockets
AJAX [Paulson05]	No	JavaScript	No	HTTP 1.1
Backbase [Backbase06]	No	BXML	Yes	HTTP 1.1
Isomorphic	No	XML based	No	HTTP 1.1

Backbase and Isomorphic are also AJAX vendors , but we classify those separately because they use AJAX technology in the background, hidden from the developer by using a declarative language. In this way, a developer does not need any AJAX or JavaScript skills to develop SPI's, because the languages abstracts the complex JavaScript and HTML DOM model [HTML]. In addition all RIA technologies provide client side XSLT transformations [Kay04], which is important to transform data sets (see Section 2.3).
Based on your requirements you can select a RIA technology. If you are not familiar with declarative programming languages you might choose Java Applets or AJAX (e.g. to speed up the development process).

3.2 Asynchronous JavaScript and XML (AJAX)

Most readers are unknown to the term AJAX and its underlying technique, therefore it will be described in this section. Asynchronous JavaScript and XML (hereafter as AJAX) is nothing new [Paulson05]. It is used as a collective noun. JavaScript is a scripting language developed in the '90 and can be interpreted by every web browser which graphical capabilities. With the coming of Microsoft's HTTP XML remote scripting component [MSDN06], we are able to load XML messages and execute JavaScript XML requests asynchronously. This technique allows us to create Single Page Interfaces, because now we don't have to refresh the whole page, but only the relevant parts of the web user interface. There are several AJAX vendors (like *Bindows*[1] or *Atlas*) and some abstract the JavaScript functions in a declarative language, such as Backbase and Isomorphic.

Backbase

This dissertation uses the Backbase RIA technology to create the web user interface. One disadvantage of AJAX is the usage of JavaScript API frameworks provided by most AJAX vendors. To build RIA applications with AJAX, you need high JavaScript skills to program your single page user interface. This is why Backbase seems to be successful, because they provide a JavaScript engine to abstract the JavaScript source by using a declarative programming language (called BXML) and thus you do not need any knowledge of JavaScript. This is why we did not label Backbase as an

[1] See more at www.ajaxian.com

AJAX technology (see Section 3.1), because it uses AJAX in the background. One other useful advantage of Backbase software is the fact that you can build your own components, inherit and reuse them. Vendors such as I*somorphic* also offers a declarative language, but do provide (custom) component creation , thus also no reuse of (custom) component, which can slow down the development process.

The Backbase Presentation Client Toolkit
Backbase offers an AJAX development toolkit, developer tools and server side support. A key element of the framework is the Backbase Presentation Client, a standards-based, ultra light JavaScript Engine that can be programmed via a declarative user interface language [Mesbah06], called BXML.

To use the Backbase RIA toolkit, a JavaScript library of 30kb needs to be downloaded during start-up. This happens just once. The JavaScript library is called Backbase Presentation Client, hereafter as BPC. The BPC is responsible for translating the Backbase Extensible Mark-up Language (BXML) into a Document Object Model (DOM), which can be interpreted by the web browser. Because BPC detects and anticipates the user's browser version (e.g. IE, Firefox), it can virtually run on any browser. This principle is also know as 'cross-browser'. Because BPC is cross-browser compliant, developers do not deal with browser issues, which also speed up the development process.

Because of the worldwide support and adoption of W3C standards, the Backbase RIA software uses the W3C standards in their BXML language, like XHTML, XML, XPATH and XSLT, which allow developers to work with familiar standards.

3.3 Conclusions

Can we use RIA to create configurable web applications? In section 2.1 we described the problems of traditional web application. We will reflect these problems to RIA and look how RIA's can solve them in the following paragraphs.

3.3.1 Data processing

In a RIA, the client is capable of more than just rendering pages, it performs computations (e.g. client-side XLST transformations), sends and retrieves data in the background asynchronously from the user's requests, redraws sections of a screen and uses audio and video in a tightly integrated manner. All independently of the server or back-end to which it is connected [Duhl03]. Therefore, RIA's seems to be very useful as the web-based client for the dashboard application, which uses large data sets and relation information to the end user [Few06].

3.3.2 Configuration and RIA

With RIA we can build FAT clients (see Section 2.2.1) which allows us to create a rich user interface environment with possibilities to create intelligent user interface wizards or agents [Dryer97]. This contributes to the creation of run-time configuration options as personalisation and internationalization.
One other feature is that RIA's can access and parse online and offline XML files and therefore could be used in online and offline applications with the ability of XML to configure the deployment settings.

3.3.3 Server Communication with RIA

The communication protocol to send/receive data packets with the server or back-end, depends on the selected RIA technology (see table 2) and stakeholders' demands. We can classify RIA communication technologies in 2 categories:

TCP Socket Connection
Technologies like Java applets, Flex, XAML and XUL use sockets connections to communicate with the server. This is the traditional communication mechanism desktop clients use to connect with a server. As developer you can decide what and how the data is sent by using socket streams.

HTTP 1.1 Connection
Because AJAX-like technologies do not need a plug-in, they still need the traditional web communication mechanism to communicate with the server. The current HTTP 1.1 [HTTP06] protocol only provide stateless key-value pairs to send to the server. But RIA's can contain intelligent logic to built a custom HTTP request before sending information to the server.
For example, Backbase uses a custom communication mechanism in the background in their server products [Mesbah06] and can send state information. But to provide user interface independence, a standard dynamic communication mechanism (like SOAP) must be implemented to enable communication with the server. Therefore this dissertation will create a small SOAP like communication mechanism to provide *user interface independence*, which can be used in every AJAX like technology. Section 6.4 will discuss this in more detail.

3.3.4 Prototyping

With most RIA technology you can build custom user interface components (see table 2), or to speed up, the standard components shipped together with the RIA distribution can be used for prototyping purposes. A major advantage is that you can change the shipped component to your needs. For example, you can customize a tree component and reuse it in all you RIA projects.

3.3.5 Testability

Because RIA's can run standalone, you can test your application/server logic independently from you web client. Nonetheless, (automatically) testing of the user interface itself still seems to be difficult [Memon02], just like other FAT client user interfaces. There is still research going on to provide a suitable solution.

3.3.6 Architecture

Rich Internet application software provides us with more possibilities on an architectural level. Because RIA's are strongly related to traditional desktop clients, we no longer need Web-, Multimedia, and Hypermedia methodologies [Preciado05] to model web applications. With RIA software, we can build FAT clients, meaning we can use traditional client/server architectures and patterns, with some technology dependent constrains like state information in AJAX applications.

4 Approach

How can we design and construct configurable web applications to visualize data relations? According to the definition, the system must be able to adapt new (data source and user interface) requirements without changing the fundamental structure of the application. This chapter describes the approach to get answers on this question and the following sub questions:

Q1. Which web technologies contribute to configurable web applications?
Q2. How can we visualize data sets and relations independent from the application logic?
Q3. Which configurability methods are available and are useful?
Q4. How do we measure configurability?
Q5. Which architectures can be used?
Q6. How can we configure variation points to adopt new requirements, without affecting the fundamental structure of the application at deployment time?
Q7. How can we configure data relations and data sets visualizations at run-time?

To answer above questions, a user interface and a framework prototype was developed and finally glued into a final pilot application, which will answer the main research question. Figure 3.1 gives an overview of the total project process.

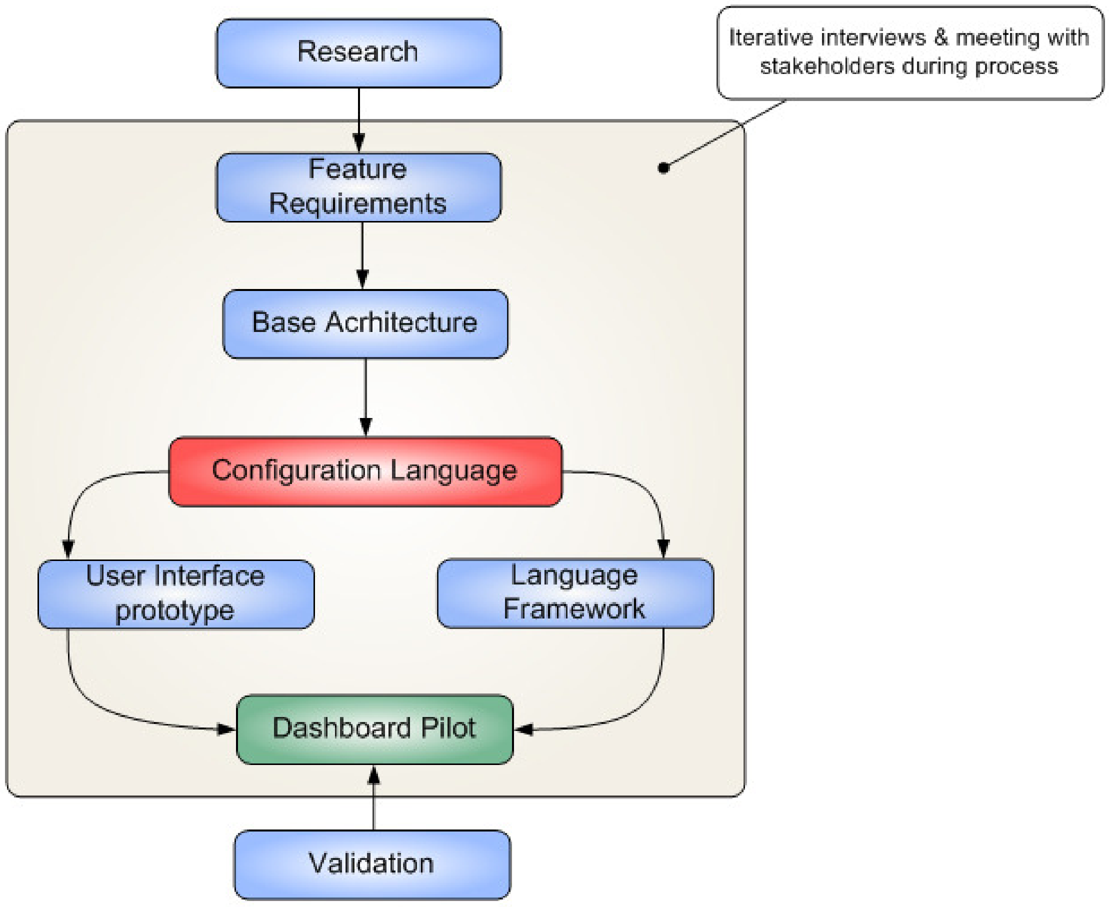


Figure 4-1, Project Process Overview

Research
First, research was done to find configurability definitions and what kind of web user interface technologies and architectures are available to create configurable web applications (Q1,Q3,Q4); this research was mainly formed by a literature study. The results can be found in the research foundations (see Chapter 2 and 3).

Feature Requirements
To identify involved stakeholders and to find out which configurability definition fulfills stakeholders demands, interviews were held [Lauesen02]. The results were documented in a context diagram with involved stakeholders and the selection of the configurability definition we used in this dissertation (see Appendix A).

Interview techniques were also used to elicit feature requirements of the pilot application, which are based on the original dashboard application. In addition, some use case diagrams were used to identify the important parts of the system (the scope of the pilot). Because of the user centric design approach, functional requirements were elicited during prototype development. We do not discuss this complete process because it is out of scope for this project. Stakeholders were involved during the complete process to get maximum feedback.

Base Architecture
An architecture was designed based on the feature requirements. Based on the feature requirements, two variability points were identified, namely variability in the user interface and data source. These variability points are the deployment variation points we want to control by using a configuration language (Q5).

Configuration Language Design
Based on the variability points indicated by the base architecture, a configuration language was designed (Q6). This XML based language is used to configure data sources and the way data set and relations are represented in the user interface (Q2). A dashboard pilot was developed to validate this language by using the configurability method proposed in section 2.7.

Prototype development
To develop the prototypes and elicit the functional requirements, we used a user centric design approach, which means stakeholders were be involved during the development process. In addition, some extreme programming best practices [Beck05] were used, like unit testing and version management. Two prototypes (language framework and RIA) were developed separately to simulate a large project where developers also could work synchronously on these parts of the application.
As in section 2.1 described, we define two types of configuration: deployment and run-time configuration. Therefore two prototypes were developed, which must use the configuration language to support the validation of user interface and data source configurability. Can these prototypes use the configuration language to support configuration without changing the fundamental structure?

Framework prototype
A framework prototype was developed to parse the configuration language and validate the user interface and data source independence.
The framework prototype affects the *deployment configuration* category; with the language the data source and user interface can be configured before run-time. Also the point where transformation (see Section 2.5) of the data sets take place could be configured via this language.

User interface prototype
A user interface prototype with RIA uses the configuration language and affects the *run-time configuration* category (Q7), so stakeholders can configure the application (only the data source) at run-time. It will also contribute to see whether RIA can be used for rapid prototyping. The Backbase RIA software was used to create the user interface prototype. How can we use the configuration language with RIA? And are we able to use the configuration language with RIA to support user interface independency, so we can make changes without changing the fundamental structure of the prototype?

Pilot application & Validation
To validate the main question of this dissertation, the two prototypes were connected with each other to create the final pilot application.
We tried to connect a different user interface (replacement of the RIA user interface) and data source by using the configuration language. We measured if the fundamental structure was changed by using the propose configurability measure method (see Section 2.7). How useful is the configuration language we designed? Based on results answers on the research questions are given. The results and evaluation are part of this dissertation, which you respectively can find in chapter 9 and 10.

5 Base architecture

The base architecture was used to identify the variability points upon the abstract configuration language was build (see chapter 6). This chapter describes the architecture design rationale and the method used to identify the variability points.

This dissertation uses the web-based dashboard application as a case study. As described earlier (see section 1.3), a dashboard application has two key elements: the data source, and the user interface to represent the data to the end-user in the form of tables and charts. To change the application without changing the fundamental structure, independence of both user interface and data source are the main stakeholder concerns. Therefore the following constraint was defined:

- User interface and data source must function independently from the application logic, so we can change the application without changing the fundamental structure of the application.

To construct a configurable web-based dashboard application, the following design decisions were made, based on the enablers that can be found in the technical background (chapter 2 and 3):

- The use of rich Internet application (RIA) technology to support dynamic configurability and the possibility of dynamic communication and client side data transformations;
- The use of a mixed client/server (smart client) model to support data visualisation rendering on both server and client side;
- The use of an off-the-shelf chart component to support visualization of data sets;
- The use of XML to create a configuration language;
- The use of XSLT to transform data set into a format as input for the chart component.

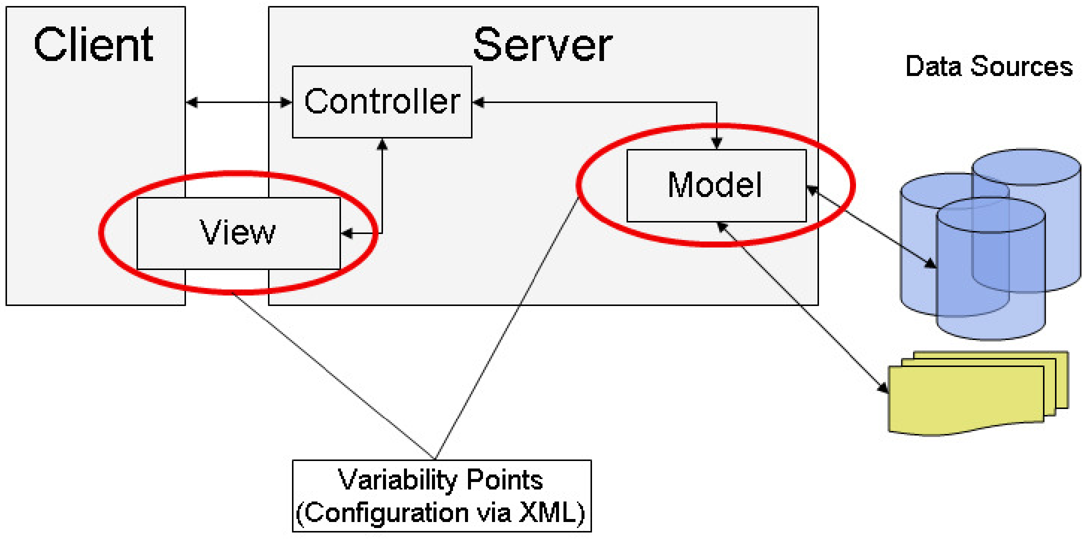


Figure 5.1, Dashboard pilot base architecture and variability points

5.1 Design rationale

To construct a configurable application, a (base) architecture needs to be designed which supports variability between those components which need to be configurable. As explained earlier, the dashboard application of this project must provide user interface and data source independency in order to change them at deployment time, without changing the fundamental structure of the application. There are many architectures (e.g. the Luther architecture [Bass03]) and design patterns (e.g. the MVC pattern) available which support a separation of the user interface and data source components, also known as presentation logic layer and data logic layer. One constraint is that the communication between the components needs to be dynamic (e.g. with SOAP, CORBA or Data Adapters).

Explanation architecture elements

Within this project the Model-View-Controller (MVC) design pattern is used to separate the user interface (View) and data source (Model) from the application logic (Controller). The 'View' is drawn in the server and client box, to indicate a mix model for the view component. Meaning the rendering of the views can take place client or server side, or both. The client box means a the web user interface and the server box is the web server.

Two variability points [Bass03] were identified (the circles in red) in the architecture design. The 'View' to support user interface independency and the 'Model' to support data source independency. Based on those two variability points a configuration language definition was developed, which we will discuss in the next chapter.

Communication from client to the controller will be done by using SOAP-like messages (see Section 2.4.1). Communication from the data source to the model is done by using data adapters and a data provider (see Section 2.4.2).

Based on the base architecture a pilot can be developed to validate the configuration language, in order to prove the system can change without changing the fundamental structure. The client and server components are the prototypes we develop individually in this project (to simulate a large project where multiple teams work onto one component).

If stakeholders have other concerns (e.g. user interface independence or configuring security settings), probable another architecture (or design pattern) might be choosen which supports separation of those specific concerns and will result into other variation points. This can also result in another language definition. This dissertation will focus on the variability points defined in figure 5.1.

5.2 Trade-off

When increasing configurability, performance will decrease. Because the information (presented by the web browser) is not mission critical, stakeholders identified performance as an not issue.
In situations where performance is an issue, stakeholders must decide which variation points need to be optimized or removed. Research on improvement of this architecture has to be performed.

5.3 Conclusion

To support configurability, an architecture design must be used, which provides variability on those components you want to make configurable. Where variable means, the component must be designed separately from others and must use a dynamic communication mechanism. Based on the identified variation points, a configuration language can be constructed.

6 XML Configuration Language

This chapter will describe the configuration language and the language framework that were constructed based on the architecture described in chapter 5 to support configuration on the variability points.

6.1 Design goals

The following main goals are defined:

- The language must support user interface independence;
- The language must support data source independence;
- The language must support the configuration of the visualization of data sets and relations.

6.2 XML Language Rationale

To support configurability on the variation points (see chapter 5) in the designed architecture, a configuration language was developed. This language abstracts the complex logic implementation details and communication between client and server and data source to increase development effort for technical stakeholders.

The language uses an XML syntax. XML facilitates the creation of new vocabularies that describe domain-specific content and context, organized into hierarchical information structures (see section 2.6). Using an XML syntax has many advantages: XML is easy to maintain, there are many free available parsers available and almost every developer is familiar with mark-up languages, because of HTML. With XML an abstract language can be provided, just like RIA does for complex JavaScript and HTML manipulation to create a single page interface (see section 2.6).

In addition, almost every FAT client-side technology can deal with XML data. Therefore, RIA software is the solution for web applications; it can read XML files very easily because of the built-in HTTP XML component in AJAX like framework, or via plug-ins (Java Applets and Flash).

The language was designed for dashboard applications, which only read data from a certain data source in order to create a visual representation. Therefore, the language now supports only reading from data sources.

The final syntax (XML schema) of the language can be found in appendix B.

6.3 XML Configuration Syntax Explained

Based on the stakeholders feature requirements, the language supports several options for configuring the dashboard application at deployment time. The language was changed several times during development and after re-factoring and feedback sessions with the stakeholders, the language evolved to what it is now.
The language contains the following main elements:

- Information about the application (information about the application, like the name)
- Resources (e.g. the physical location of XSLT style sheets)
- Targets (the user interface which can connect to the application logic)
- Transformers (references to XSLT transformations to configure the way data sets are visualized)

- Data sources (available data sources and credentials)
- Data set filters (filters the data out of the defined data sources, the result is a custom data set)

The most relevant elements of the XML configuration file will be discussed in the next paragraphs via an example.

6.3.1 The Benchmark Example

To explain the language, a benchmark utility to compare metrics results from different software systems will be used as example. The benchmark utility uses an MySQL database and Hibernate to persist the data as Java objects collection. The database can be queried via hibernate queries and uses a web user interface to represent the results. Figure 6-1 illustrates the simplified database entity-relationship diagram (ERD).

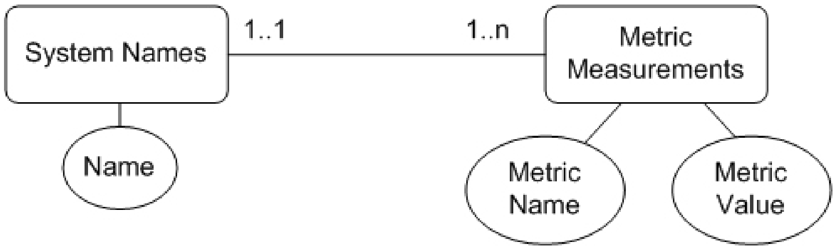


Figure 6-1, Benchmark Example ERD

Each system in the database has a system name. And each system has one or more measurements (like total lines of source code of the entire system).
In addition an RIA web user interface will be used to configure the data sets and how they will be visually represented. Figure 6-2 is a screenshot from the RIA user interface.

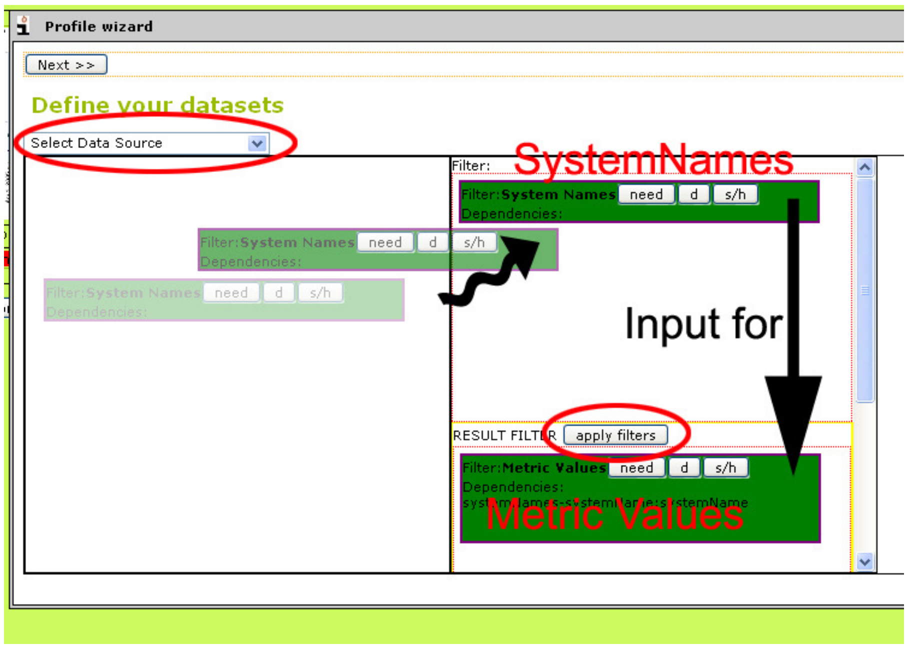
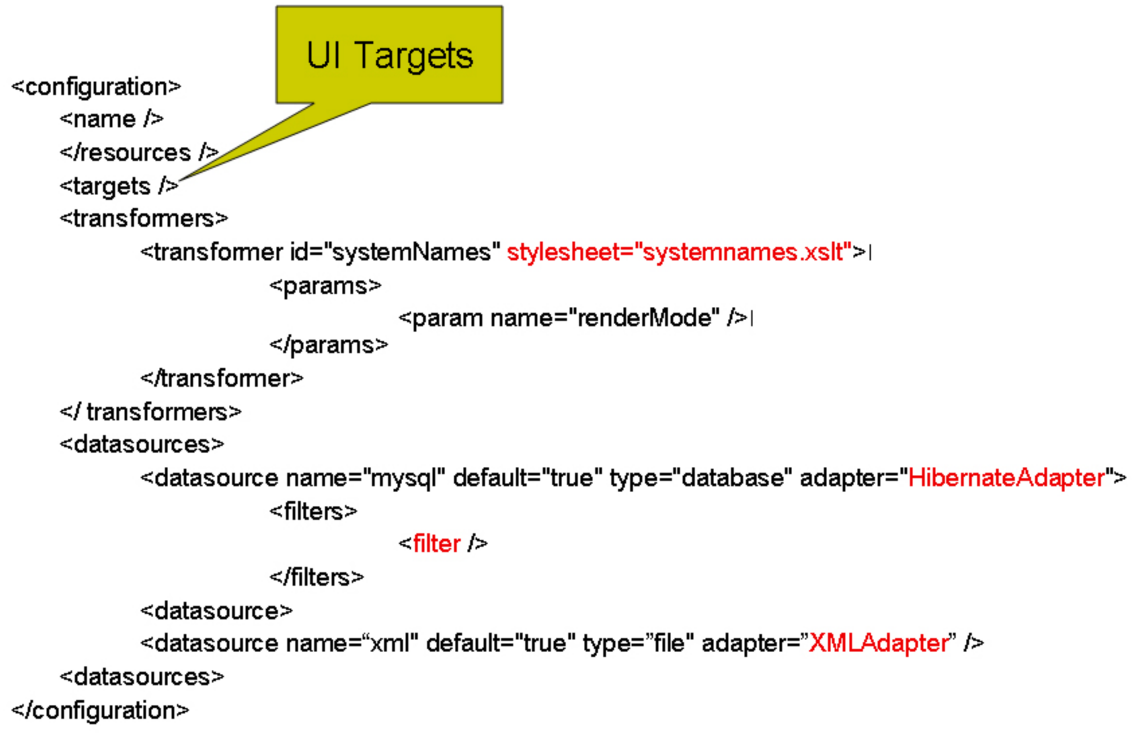


Figure 6-2, Select data source and define data set and relation

Figure 6-3 shows how the configuration syntax looks when we use the setting described above.



```
<configuration>
    <name />
    </resources />
    <targets />
    <transformers>
        <transformer id="systemNames" stylesheet="systemnames.xslt">
            <params>
                <param name="renderMode" />
            </params>
        </transformer>
    </ transformers>
    <datasources>
        <datasource name="mysql" default="true" type="database" adapter="HibernateAdapter">
            <filters>
                <filter />
            </filters>
        <datasource>
        <datasource name="xml" default="true" type="file" adapter="XMLAdapter" />
    <datasources>
</configuration>
```

Figure 6-3, Language syntax overview

6.3.2 Resources

The resource element in the configuration language was designed to store information about external resources. For the dashboard pilot the location of the XSLT stylesheets and the physical location of the charts components could be specified.

6.3.3 Targets

The 'targets' element introduce categories of transformations (categories for each user interface target) and the location where these transformation can take place (on the server or on the client). Based on your target the framework (see Section 6.4) will automatically selects, for example, the RIA XSLT transformations to render the user interface parts requested by the client. In this way, you can configure how data sets will be transformed and where. In section 6.4 the workflow will be explained in detail.

6.3.4 Transformers

Transformers are the elements where you can specify one or multiple XSLT transformations and what parameters this XSLT expects. Each transformer has his own unique identifier, so we can target the transformers. In figure 6-3 you can see a transformer with a name called 'systemNames'. This transformer needs one XSLT parameter ('renderMode') in order to transform the XML source into multiple formats (like a transformation to a dropdown box or as chart data).

6.3.5 Data sources

The data source element can contain one or multiple data sources. Each data source element has a name, type and adapter name. Figure 6-3 shows different data sources (*mysql* and *xml*) and the related adapters (*HibernateAdapter* and *XMLAdapter*) as attributes of the data source element.
In addition, you could also specify extra data source credentials, like location of the data source, username and password etc.
The data source element can contain multiple filters. Filters are a specific selections of data from the data source.

6.3.6 Data set filters

A data set filter is a subset of data out of the data source. Each filter has a name and unique identifier to target them. Notice that the example use the filter names in the RIA user interface, see Figure 6-2. This is because the client also can access the configuration file (see Section 3.3.2). A data set filter contain the following elements:

- Query
- Output format
- Parameters
- Dependencies

Query
You can specify a data subset by using a query; SQL in the case of a database, XPath or XQuery [Kay04] in the case of XML, ranges of cells in the case of a spreadsheet or some other query language specific for the data source. The results could be either a collection of Java Objects or XML. It now depends on the adapter you specify for the data source, but could be customized in the future.

Parameters
Some data source queries require input parameters. Like a system name to get all the related measurements (see Section 6.3.1, the Benchmark example). These parameters can be specified in the parameter element.

```
<params>
   <param name="systemName" type="xs:string" />
</params>
```

Dependencies
It could also happen that a certain filter is dependent from another data set filter defined in the configuration. Or in above example, the *systemName* is the input parameter of another query, to get the related measurements. Those dependencies could be defined in a 'dependency' element.

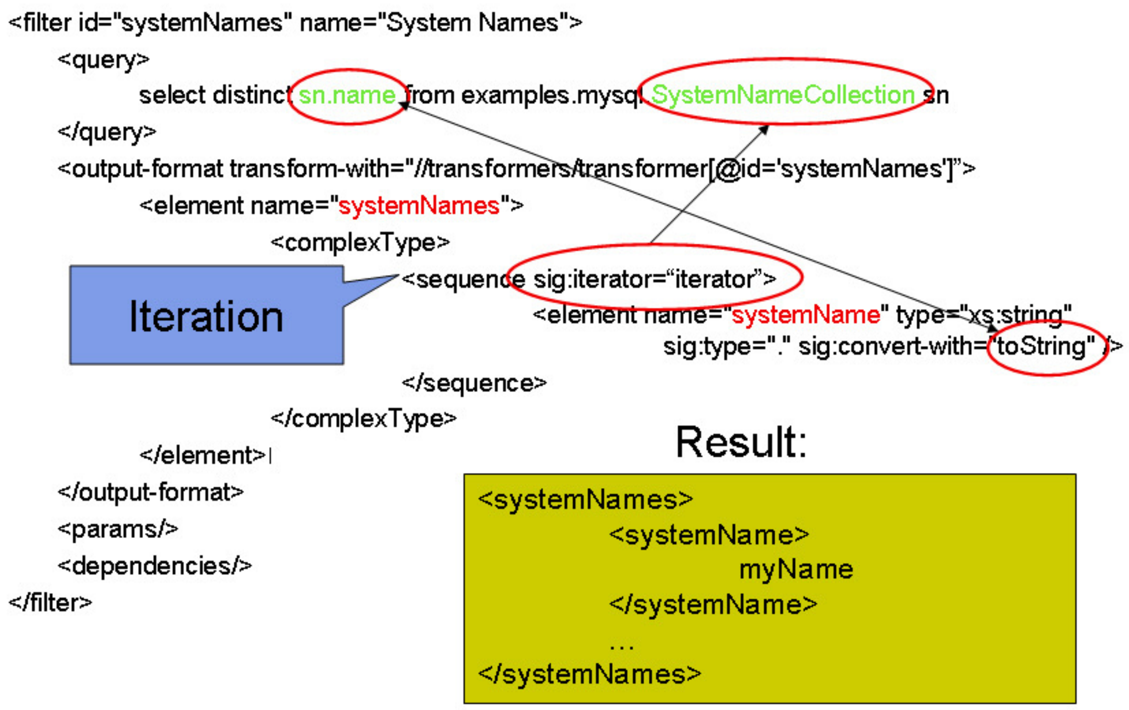
```
<dependencies>
   <dependency filterID="systemNames"
   filter="//datasource[@name='mysql']/filters/filter[@id='systemNames']" >
      <mapping map-element="systemNames" map-to-param="systemName" />
   </dependency>
</dependencies>
```

You can specify an XPath query in the 'filter' attribute to point the dependent filter and how to map the results from filter A to filter B by using the 'mapping' attribute. In above example, the 'systemNames' attribute will be mapped to the systemName in order to get measurements out of the data source, based on the system name (see Section 6.3.1, the Benchmark example).

Because a developer knows what the results of certain queries are, he also could transform them. You can completely specify the output format of the filter in XML by the use of XML schema (XSD) [Harold04]. How this works will be explained in the next paragraph.

6.3.7 Output format

With the output format element the developer can map the result from the query (see previous paragraph) to a XML format by using the W3C XML Schema language, also known as XSD. To map the result, an example is provided (see Figure 6-4). This example maps Java Objects (the format the framework returns) to a custom defined XML format. We use the measurement collection from the example.



```
<filter id="systemNames" name="System Names">
    <query>
        select distinct sn.name from examples.mysql.SystemNameCollection sn
    </query>
    <output-format transform-with="//transformers/transformer[@id='systemNames']”>
        <element name="systemNames">
            <complexType>
                <sequence sig:iterator="iterator”>
                    <element name="systemName" type="xs:string"
                        sig:type="." sig:convert-with="toString" />
                </sequence>
            </complexType>
        </element>
    </output-format>
    <params/>
    <dependencies/>
</filter>
```

```
<systemNames>
        <systemName>
                myName
        </systemName>
        ...
</systemNames>
```

Figure 6-4, Output format

If the result is a Java object, you could iterate the ‘SystemNameCollection’ by using the XSD *sequence* element. You could specify the iterator type to iterate over an collection (ArrayList, Hashtable etc.) by using the ‘sig:iterator’ attribute. Notice the XML namespace ‘sig:’ to separate the XSD attributes from the configuration language attributes. The iteration in figure 6-4 will give a list of Java objects, which will be translated into XML elements. The properties of each Java object in the collection can be translated by giving a method name of that object (in figure 6-4 the ‘toString’ method is used). Figure 6-4 shows the iteration over the system names collection and the result of the mapping.

Finally the ‘transform-with’ attribute of the output format element, is using an XPath query to target a transformer in the configuration file itself. In this way data can be transformed to visually show the data set (could be a transformation to a table or to a chart XML format etc.). Notice this can be done, without changing any Java source code.

Because a developer can specify his own return format, he also knows how he can handle this format, for example, to transform it with a XSLT transformation. If a developer does not use a server-side transformer (read server-side XSLT transformation), he can get the raw XML data and transform the result client-side. Therefore, this technique supports the smart model mentioned in section 2.2.3.

6.4 Configuration Language Framework

In order to parse and use the language (semantic), a framework was developed. This framework can read the XML configuration language and provides several interfaces to connect with the user interface and data sources.

6.4.1 Development process

To construct the pilot framework, the extreme programming methodology [Beck05] was used. Starting with the easiest part and extent the source code every (weekly) iteration to construct the final framework prototype. The following iterations were defined:

- The language parser, to read the configuration language
- MVC Controller construction, the application logic to send and receive messages to the client(s).
- MVC Model construction, data adapters and data provider and the mapping with the language parser.
- MVC Views, the XSLT transformations to transform the data to the chart component input format

6.4.2 Parsing and Reading the XML Configuration

To explain the process of parsing and reading the configuration language (language semantic), an example of the workflow process, a request to get all system names from a MySQL database and updating the user interface part is provided. Figure 6-5 illustrates the workflow process for this example and shows how the framework works.

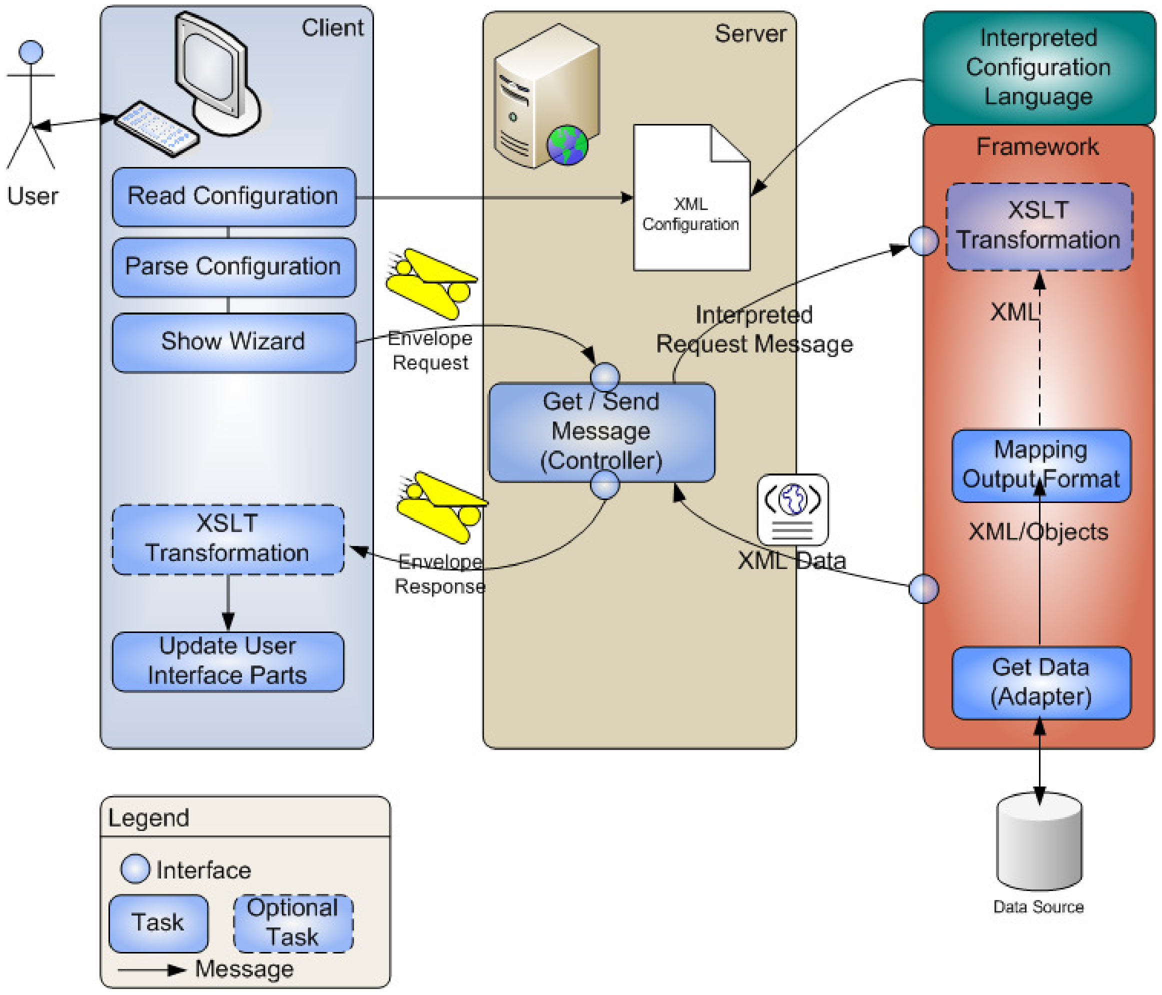


Figure 6-5, Workflow process, from client request to server response

Explanation:
The *configuration* will be *read* and *parsed* by the client. Based on the entries in the configuration file (like available data sources and data set filters), the client will render a *wizard* (see Section 7.3). When a user does a data request (by using the wizard), an XML message is send to the server with parameters like the unique filter identifier and the target user interface. The server controller handles the XML message and will send it to the configuration language framework. The framework will also read and interpret the configuration and get the data, based on the unique filter identifier and client target. Optionally, this data would be transformed by using an XSLT style sheet, defined in the configuration and based on a given client target. The controller will receive XML data back from the framework. The data will be send to the client (optional transformed by client XSLT) and the affecting user interface parts will be updated.

All internal data communication between the components are generic because of the use of XML. The dynamic communication mechanism, developed for sending messages to and from the server via the user interface (XML envelope), is the key to create user interface independent web user interface. Therefore the proposed technique will be described in more detail in the next paragraph.

6.4.3 Client/Server communication

Because we are using a zero-install (AJAX) RIA technology, meaning we have to build our own communication protocol if we want communication with the server other then POST and GET messages (see section 3.3.2). Other then SPIAR [Mesbah06], we suggest a communication protocol that is more flexible. This can not only used by Backbase technology but also by other AJAX technologies, which leads to more widely community support.

To create a dynamic communication protocol between client and server framework we need a standard (more justification). Web services uses the SOAP protocol to handle communication. Because of the short duration of this project, we need something simpler in order to implement it for the dashboard pilot.

Backbase has a communication mechanism in their server family product. It uses a small set of Base64 encoded instruction list to send state information from client to server and vice versa. These messages can contain user interface style, behavior and structure information (see Section 2.4.2). However, this technique is hidden in their Backbase's framework [Mesbah06] and can only be used in a .NET or J2EE architecture and therefore no support for custom JSP, PHP or other techniques is provided. If we want to create user interface independent web applications, we need a protocol, which can be used in every user interface and back-end technology. So why not using today's standards?

With traditional web applications all communication is done via HTTP POST or GET methods. Our suggested XML envelope uses XML HTTP requests to send and receive messages.
The client builds a envelope message to send back to the server. The server will receive the envelope and send a envelope response back to the client, based on the client's envelope request message.
We only used it to request XML data sets based on several parameters (like the name of a filter and the render mode), but you could also think of events which are occur in the client, to keep the server in sync.

```
<sig:envelope>
  <sig:url />
  <sig:parameters>
    <sig:param name="ClientTarget">RIA</sig:param>
    <sig:param name="FilterID">systemNames</sig:param>
    <sig:param name="RenderMode">AsChart</sig:param>
  </sig:parameters>
</sig:evelope>
```

The advantage of this technique is that we only need one entry point in the framework, which translate the envelope message into an intern format. In this way, also other clients, which send these XML envelope messages, can access the framework and get responses. This is also, how application logic uses SOAP web services. Therefore future work could be connecting RIA directly with web services by using the SOAP protocol.
This communication mechanism is validated by creating a command line user interface (see chapter 8, the dashboard pilot), which connects to the same framework entry point. We did not implement complete user interface state information, other then selected filter options, because we did not need more state information for the dashboard pilot.

6.5 Alternatives

As an alternative solution, the defined variation points in the architecture could also be configured by using some Java classes. But then stakeholder, like system administrators, need to change the source code (like setting database credentials) and recompile the application.
One other idea that was considered, was using the Java Spring framework [Spring06] which also allow developers to configure application components by using a XML based configuration language. On disadvantage of this technique is that you can't specify you data output format (see Section 6.3.7) and that the clients cannot access these configuration files directly (e.g. to gather available data source names), because they first need to be interpreted by the server, which in our case was needed in order to configure the user interface at deployment time.

6.6 Discussion And Future Work

The language was used for the pilot dashboard application and therefore only uses elements specific for this type of applications. In order to use it for commercial purposes and other applications, more research needs to be done, like storing data into the data sources by using the output format as a mapping.
The configuration language is still in prototype phase. This project was using two variation points upon which the language was built. Therefore, more research must be done in able to refine the syntax to support more variation points.
Based on the configuration language definition (XSD) we can generate an framework API to read the XML configuration language syntax. Framework tools like JAXB [JAXB06] can parse an XSD schema and automatically generate a Java API library. This could be useful to adopt new changes in the syntax (e.g. support of other variation points). Because of the short duration of this project a small custom framework was developed.

Future research must be done on deployment configuration. Also the abilities of implementing a complete SOAP protocol in RIA applications, so we can communicative with web services and hold complete state information of the user interface. Once this can be realized we can think about creating distributed user interface components via web services.

6.7 Conclusion

To configure a application at deployment time, first an architecture must be designed, which separates the components that must be configurable (in this project the user interface and data source). One constraint is the use of an dynamic communication mechanism, like SOAP or data adapters. Based on the variation points, a configuration language can be defined. The language proposed in this dissertation will be parsed by using a framework (or API). This framework provides several interfaces, so the final application logic can connect with it.

7 User Interface Prototype

The main purpose of the user interface prototype is to elicit and validate requirements in terms of the run-time configurability. As described before (see Section 1.5) configurability affects the customization to visualize data set and data relations. This chapter will answer the following sub-research questions:

- Which web technologies contribute to configurable web applications?
- How can we visualize data sets and relations?
- How can we configure data relations and data sets visualizations at run-time?

The following user interface prototype objectives were defined:

- Select RIA technology;
- Develop the user interface prototype by using a user centric design approach and with the selected RIA technology;
- Describe how RIA can use the configuration language to visualize data sets and relations;
- Describe how RIA contribute to run-time configuration.

7.1 Selecting the RIA technology

Based on your (feature) requirements you can select a RIA technology (see section 3.1). This project uses Backbase technology, because of the following stakeholder concerns:

- The organisation, within this project is executed, does not want to install browser plug-ins for security reasons;
- The declarative XML based language (BXML) provided is based on W3C standards and can be mixed with HTML. Therefore, developers familiar with HTML can easily speed up the application development with this language;
- Reuse of components is supported, so rapid prototyping is supported.

7.2 Development Process – A User Centric Design

The description of the development process is one of the objectives of this dissertation and does not contribute to answering one of the research questions. Therefore this paragraph could be skipped for those readers who are not interested in the user interface development process.

The user interface prototype was developed using a user centric approach [Cooper03]. With this approach, stakeholders are involved during the development process and stakeholders will see (visually) progress on a short term, which elicits feedback from the beginning of the project until the end. This feedback and results of this prototype were translated into requirements.

When you develop by using a user centric design, user involvement is the most important thing. One way to get user involvement is to create wireframes [Snyder03]. Wireframes are prototypes and can be created either offline or online. Offline means just on paper (like a story board) and online means a clickable application with a minimum of application logic. With wireframes you can also gather functional requirements. In this chapter, we discuss the offline and online wireframes, which were developed for the prototype and will answer whether RIA technology is useful to rapidly create prototypes.

7.2.1 Offline wireframes

The goal of offline wireframes (or paper prototyping) is to get the basic graphical user interface lay-out and the functional requirements [Snyder03]. By using the feature requirements, the offline wireframes were created. Our main goal was to develop a graphical user interface, which could fit onto the screen [Few06] and which you can control by a minimal number of mouse clicks. Therefore, we did not use scrollbar controls or others, which could disturb the "fit at screen" principle.

The wireframes were presented to the stakeholders. Since this is an iterative process, the wireframes changed until stakeholders were satisfied. The following wireframe description is the result.
Before you start drawing wireframes it is useful to create some basic use case diagrams to created a overview of functionalities. Based on the use cases of the dashboard application of this project, stakeholders defined two possible screens. The first is a default screen, which is visible for every user of the application. The second is the screen administrators can see when they are logged in -the run-time configuration mode, also known as the customization mode (see Section 2.5). In this view users can customize their settings. They can choose either they want to change the current layout (e.g. where would they like to put the charts) or to customize the data sets and type of data set visualization (e.g. a table or a pie chart). To change the layout they see a window where they can choose a different dashboard lay-out. To change the data sets, the use enters a two-step wizard. Step one; select data source and select data filters; step two; select a chart or table lay-out. All above described tasks are defined into wireframes. Figure 7-1 is an example of the wizard.

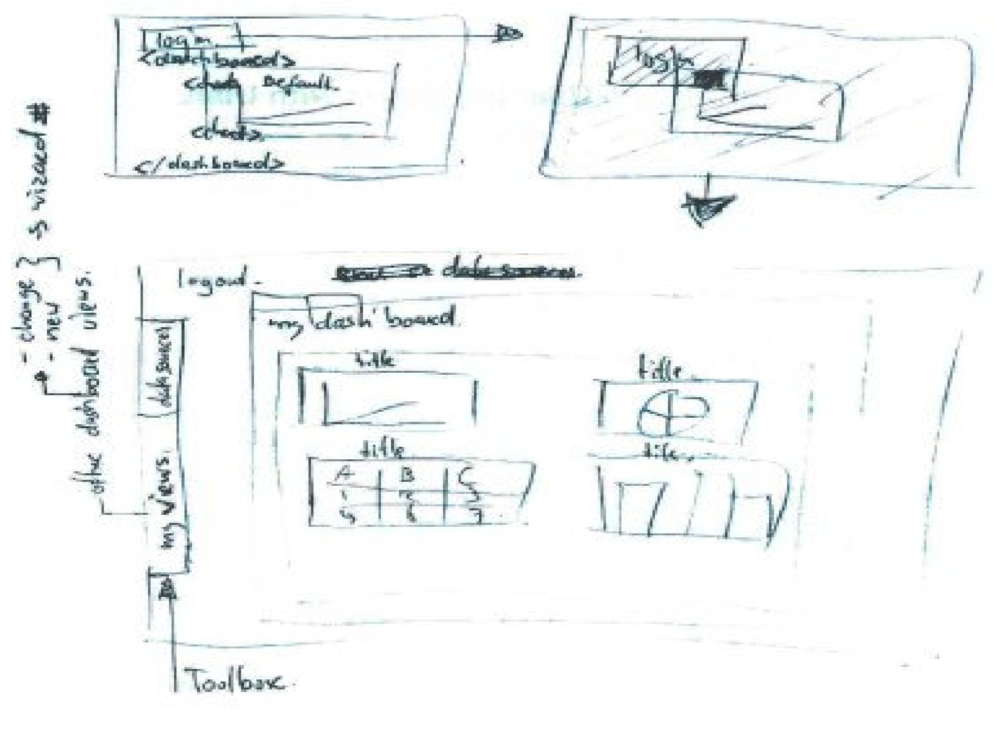


Figure 7-1, Offline wireframes of the dashboard customization mode

7.2.2 Online wireframes

Online wireframes are a clickable prototype of the application. There is no application logic and no real data used. The purpose of online wireframes is to get an impression of the final application. Most interaction designers use MS Powerpoint to create wireframes, but those aren't clickable. With online wireframes stakeholder can actually click and can feel the interactive experience. Based on this experience they can say if they like it or if it has to be changed. If they want to change something about the user experience, then you have to back to the drawing board and draw the offline wireframes as we discussed in the previous paragraph. It's much easer when you already have offline wireframes.

The online wireframes are developed by using Backbase's BXML. One benefit of using BXML is the standard user interface controls they deliver together with the free toolkit. Based on the offline wireframes we made a selection of default user interface controls.

We discussed the online wireframes with all stakeholders in a stakeholder meeting [Lauesen02]. By feeling the new user experience and with their knowledge about the domain, they came up with new feature requirements. One of features was a breadcrumb functionality. With a breadcrumb function they can see where they are in a certain workflow [IDDesign06].
We also had some end-user sessions. Within these sessions, the user could click though the application. Features about drilling down in charts and data set visualization in the wizard were results of those sessions.

After finalizing the online wireframe we can reuse some user controls and create our own controls to use in the final pilot. The wireframes are only front-end code, defined in XML and BXML source files.

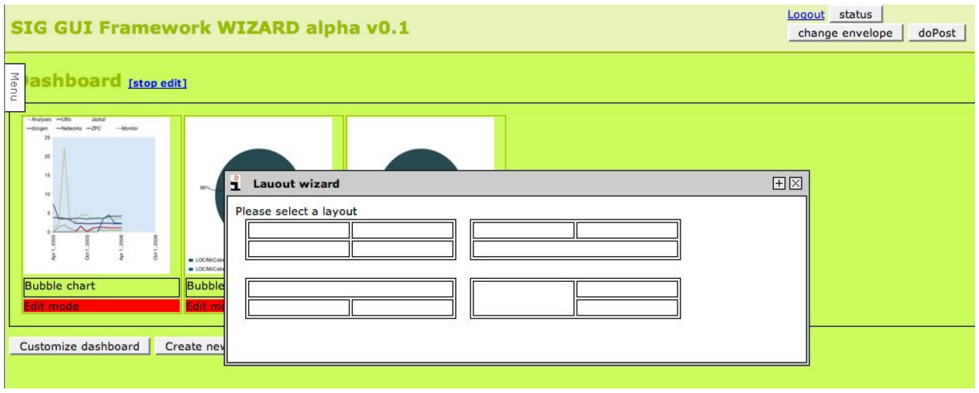


Figure 7-2, Screenshot of online wireframes, the customization of the lay-out

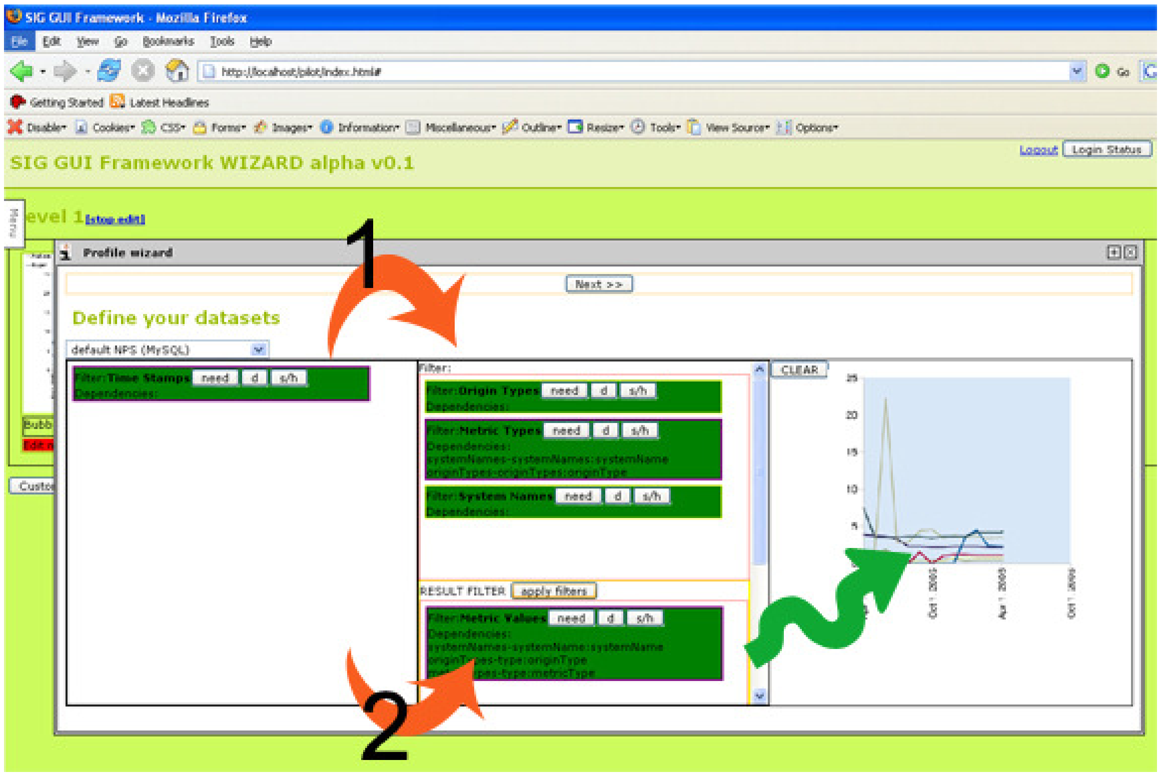
7.3 Visualisation of data sets

This paragraph will answer the question of how to visualize data set and relations and what role RIA plays to support run-time configuration. The visualisation and customization of data sets is part of the user interface prototype. We defined a separate prototype to create a wizard user interface component. The wizard component was reused in the final user interface pilot.

7.3.1 Data set and relation wizard

The data set wizard is an intelligent wizard [Dryer97] which is able to read the XML configuration language directly (see Chapter 5) from a given file location.

Information of available data source and filters can read in this way (see chapter 6). By using client side XLST parsing (see Section 2.5), some parts of the XML configuration are transformed into a visual representation [Tsoulos03] in the web browser (see figure 7-3)



SIG GUI Framework - Mozilla Firefox
http://localhost/pilot/index.html#
SIG GUI Framework WIZARD alpha v0.1
Logout
Login Status
evel 1 [stop edit]
Profile wizard
Next >>
Define your datasets
default NPS (MySQL)
Filter:Time Stamps need d s/h
Dependencies:
Filter:
Filter:Origin Types need d s/h
Dependencies:
Filter:Metric Types need d s/h
Dependencies:
systemNames-systemNames:systemName
originTypes-originTypes:originType
Filter:System Names need d s/h
Dependencies:
RESULT FILTER apply filters
Filter:Metric Values need d s/h
Dependencies:
systemNames-systemName:systemName
originTypes-type:originType
metricTypes-type:metricType
CLEAR
1
2

Figure 7-3, Step 1 and 2 of the data set wizard

Based on the configuration file, filters are present visually. Stakeholders can drag-and-drop the filters (which give nice visual capabilities to the end-users) to create a new data set. To create a data set two (optional three) steps are needed:

1. (optional)Relate exciting filters to build a data relation (selection of the filters to get data sets as input for one other filter, if there are dependencies).
2. Select the filter to get the final data set
3. Select the way the data set can be visualized (see Figure 7-4).

An example: Filter C needs a name and address as input in order to get a data set of telephone numbers. Filter A distillate names from a data source and filter B addresses. So, filter C needs A and B in order to get a data set by name and address. Step 1 in figure 7-3 shows three dependent filters for the final data set (step 2). The presentation logic automatically detects dependencies and will show the user when an error occurred. For example, if filter C depends on A and B, filter C cannot be used, before A and B are selected. The user interface automatically highlights the dependent filters once selected. Once the final filter is set with all right dependencies, the user can apply the filter (see step 2 in figure 7-3).

After applying the filter and selecting the data visualization (step three), the client sends the user settings to the server. The results will be stored in a (personalized) folder and could be transformed as plain text, or as charts. A developer can control where data sets are transformed, either on the client, by using client side XSLT transformation, or server side (see Section 6.3.3).

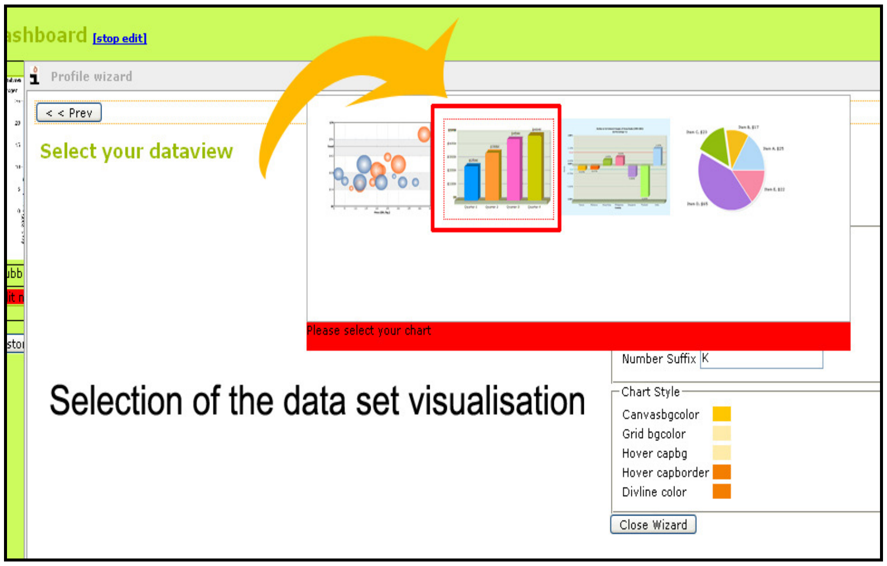


Figure 7-4, Step 3 of the wizard - Select data set visualization

7.3.2 Selection of COTS components

With the data set wizard, we are able to generate XML data sets. Developing data set visualisations is a very time-consuming process and there are a lot off-the-self components available on the web. Freely available or with a minimum of costs. The original dashboard monitor tool uses KavaChart [KavaChart06] and uses JSP technology to generate the charts as an image.
Because our framework uses XML to present data set, we need a component, which can deal with XML data sets. This is our main requirement. In addition, we need drill-down possibilities (future requirements).
We select the off-the-shelf components by looking at the following features:

- XML based to communicate with the language framework;
- Drill down possibilities;
- Animated charts for customer presentation purpose.

We selected the FusionChart [FusionCharts06] component, because it supports our defined requirements. Figure 7-5 shows an example of a data set visualization.

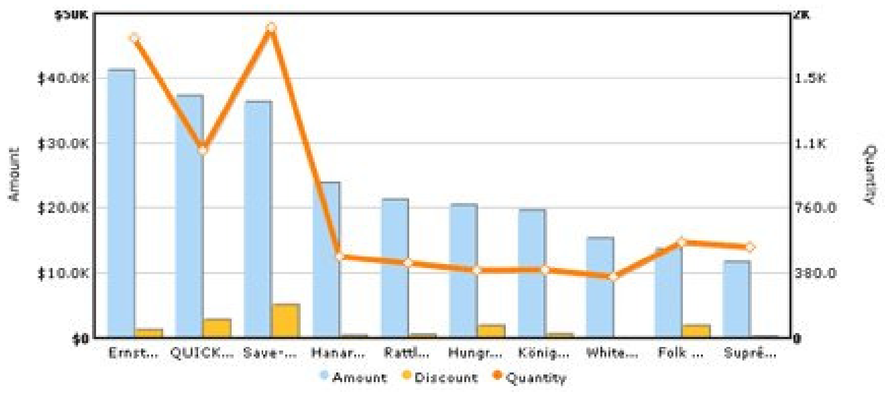


Figure 7-5, A example of results in the Fusion Chart 3D bar chart component

The custom defined XML data sets are transformed server-side (server in control) to the chart component XML format. In addition, client side rendering is possible here, but because the data sets could be large, server side is (for now) much faster.

7.4 Reuse of interface components

One benefit of Backbase RIA software is the possibility of reusing the user interface component developed for the prototype. The component could be included like C++ header files in the user interface source code.

```
<s:include b:url="controls/b-window/b-window.xml" />
```

Example, inclusion of a window control in BXML

The controls are defined in the BXML programming language. The components are stored in separate XML files on the server. But could also stored at another location on the internet. Another web server of web service can also provide user interface components. This gives us the ability to include *distributed user interface components*, which we can configured with the configuration language in future.

A RIA user interface component is build out of four main elements:

- Behavior (how the component acts and uses user events, for example, how to process a user's mouse click)
- Structure (how the components fits into the internal object browser tree, called Document Object Model, also known as DOM)
- Style (how the component is represented to the user in the browser)
- Content (the text/data the component might be contain)

Because a user interface component can contain the above characteristics, it can be function autonomously, which improve reuse and can speed up the user interface development process and reduce development costs.

7.5 Conclusion

The user centric approach helped to rapidly create a user interface prototype. Wireframes seem to be a useful technique to elicit functional requirements. Thanks to the Backbase RIA out-of-the-box controls, we were are able to create a clickable prototype in just one week. The user interface components can be reused in the final pilot as well because the can function autonomously.

With RIA technology you can easily access XML data sources from any location on the internet. Therefore RIA software is very useful technique to read the designed configuration language in order to create visualisations of data sets and relations.

Because of the rich possibilities of RIA, we can create 'intelligent' wizards. This will allow us to create wizards to customize data sets and relations and the way they will be visualized. Off-the-self chart components can help us to visualize the customized data sets and reduce development time and costs.

8 The Dashboard Pilot

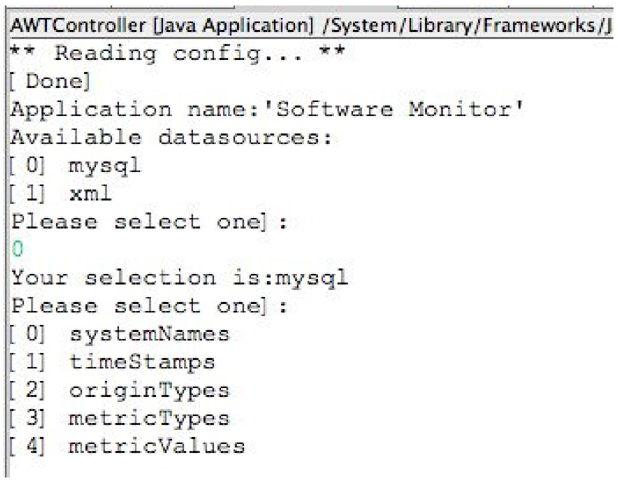
The developed prototypes (the language framework and the RIA, see Chapter 6 and 7) were glued into the final dashboard pilot, to support the validation of the main research question. If the pilot must contribute to the definition of configurability, we must be able to change the application without changing the fundamental structure of the application. Because the SIG organisation requirements were data source and user interface independency we validate those feature requirements by using the validation method proposed in section 2.7.
We defined the following objectives:

- Define the fundamental structure of the application.
- Replace the MySQL data source with a XML data source and measure the impact on the fundament structure of the application.
- Replace the web user interface with a console line user interface and measure the impact on the fundament structure of the application.

Fundamental structure
To define the fundamental structure of the application we looked at the base architecture (see Chapter 5) to identify the non-variable components. In our case this was the MVC Controller component (see Figure 5.1) and the framework logic to parse the configuration language. The total file count of the fundamental structure includes 32 Java source files in 7 directories. More details can be found in table 3.

Data source replacement
The current MySQL data source was replaced by an XML data source. To add a XML data source, we only need to write a XML adaptor, which implements an interface provided by the configuration framework. We added a new section (element) to the configuration file and defined new (XPath) queries in order to get data sets out of the XML data source. After changing the data source, the affected source files within the fundamental structure were counted. The result was 0 (see table 3).

User interface replacement
To validate user interface independency we created a console line client and implemented the same dynamic communication mechanism as in section 6.5 described. Afterwards we change the client targets (Section 6.3.2) in the configuration language to detach the RIA user interface and replaced it with the command line user interface. The console line application (figure 8-1) also used the same wizard as in section 7.3 described and also uses the configuration language to select the data filters and relations, but without showing them in a visual representation (like a chart), but only the pure response data. We also considered a Java Swing user interface, but because of the short duration of the project this was planned for future work. We counted the affected source files within the fundamental structure after implementing the console line client. The result was 0 (see table 3).



```
AWTController [Java Application] /System/Library/Frameworks/J
** Reading config... **
[ Done]
Application name:'Software Monitor'
Available datasources:
[ 0]  mysql
[ 1]  xml
Please select one] :
0
Your selection is:mysql
Please select one] :
[ 0]  systemNames
[ 1]  timeStamps
[ 2]  originTypes
[ 3]  metricTypes
[ 4]  metricValues
```

Figure 8-1, Console user interface to prove UI independency

The table below gives an indication of the total development effort needed in order to add new data source and user interface requirements. Notice the few lines of code (LOC) we added to the configuration file. The added lines of code are for one data set filter (see Section 6.3.6, data set filters).

Table 3, Result after adding new requirements

Artifact	Before (*1)		After (*2)		Total
	Total LOC		LOC Added		Total LOC
User interface Layer	2730		198		2928
Data source Layer	292		133		425
Fundamental Structure	**2930**		**0**		**2930**
Configuration File	226		11		237
	File Count		Files Added		Total File Count
User interface Layer	22		1		23
Data source Layer	2		3		3
Fundamental Structure	**32**		**0**		**32**
Configuration File	1		0		1
1: With 1 Data source (MySQL) and 1 user interface target (RIA) 2: Added XML data source and console user interface					

8.1 Conclusion

By using the configuration language a new data source and user interface was added to the dashboard pilot application. The proposed validation method in section 2.7 was used to measure the configurability. The result were positive because the pilot application was able to successful adopt new user interface and data source requirements, without changing the fundamental structure of the application.

9 Results and Analysis

This chapter answers the main and sub research questions (see Section 1.6, Q1-Q7). The main research question was: "*How can we create configurable web applications to visualize data relations?*". The answer to this question is twofold:

- A abstract configuration language, in combination with;
- Rich Internet Application software.

In the following sections the results and analysis are described. Finally, a discussion about configuration and programming, future work and conclusions are described.

9.1 Abstract Configuration Language

To configure an application at deployment time, first, an architecture must be designed which separates the components that must be configurable (**Q5**). In this project we identified two variation points, the user interface and the data source component (see Chapter 5).
Based on the variation points in the architecture, an XML based configuration language (**Q3**) can be defined (see Chapter 6) as an abstraction of the underlying complexity of the configurable components (**Q6**). One benefit of an XML based language is that it is easy to maintain the grammar while the configuration language is developed.

The domain specific language is interpreted by using a (custom) framework (Section 6.4). This framework provides several interfaces, so the application logic (fundamental structure) can connect with them. For example, to receive user messages from the user interface. One constraint is the use of a dynamic communication mechanism, like SOAP or CORBA, to create a dynamic coupling between the configurable components and the application logic (the fundamental structure).

The configuration language enables configurable data sets, data relations and user interface targets (see Section 2.4) for the dashboard pilot, without changing and recompiling the fundamental structure of the application.

9.1.1 Discussion - Alternative solutions

As an alternative solution, the defined variation points in the architecture can also be configured by using Java classes or the Java Spring Framework (see Section 6.5). In this way, stakeholders (like system administrators) need to change the source code (like setting database credentials) and recompile the application (see Section 6.5), which was not acceptable for the SIG organisation.

9.2 Rich Internet applications

Configurable web applications can be created by using RIA (**Q1**). Because a RIA can access, read and transform XML files from any internet location, it can read the XML configuration language that is provided in this dissertation and use it to provide run-time configuration options to the end-users (Chapter 7). For example, by using a wizard (see Section 7.3.1).
In addition, desktop like clients can be build with RIA technology. This gives more possibilities in design and user interaction, like building an intelligent wizard to customize (**Q7**) and visualize (**Q2)** data sets and relations.

Because RIA provides almost the same user interface possibilities as traditional desktop client, such as rich controls and asynchronously loading of data, the user experience can be improved. Traditional architectures can be used, rather then web architectures. But because of the current internet speed, some tasks still needs to run on the server (for example complex data set transformations). In addition, communication between client and server is not yet standardized. Therefore, a custom SOAP-like, communication mechanism was developed (Section 6.4.3). With this custom communication mechanism, we were able to send small messages asynchronously from the web client to the server and visa versa. For example, the server could update the user interface wizard (see Section 7.3.1) without any page refresh (see Section 3.2). This is impossible to achieve with traditional web applications (see Section 2.1).

More advanced communication protocols (see Section 6.4.3) can be introduced when using RIA, while the traditional and widely available HTTP protocol can still be used. This can give a new boost to internet application development. Now, it is possible to send more complex data structures to the server and read configuration files. With these abilities, traditional desktop client/server architectures, which can create user interface independent web applications, can be used.
Because most RIA software technologies (except for Java Applets and AJAX) provide a declarative user interface language (Section 3.1), which introduces a higher abstraction level of user interface definition, it is possible to rapidly develop prototypes (see Chapter 7).
Most RIA languages provide a component based architecture and therefore the developed user interface components from prototypes within the final software system can be reused, which increase development speed (see Section 7.4).

9.3 Analysis

For our analysis we used the results of the dashboard pilot we developed for this project (see Chapter 8). We implemented new user interface and data source requirements in the pilot application by using the configuration language. According to the configurability definition (see Section 1.5), the pilot application is configurable because we were able to implement these new requirements without changing the fundamental structure of the application (see Chapter 8). We proved this by using the validation method described in section 2.7 (**Q4**).
In our case study, the application was completely configurable via the configuration language. However, with this method we can only determine if the application is completely configurable or not (true of false), see Section 2.7.
The method is based on the configurability definition by [Sommerville06], and the assumption that the application can be clearly separated into configurable components and a fixed fundamental structure, like in our case study. But what if we need to change two files in the fundamental structure? Then, according to our method, the application is not configurable. This problem never occurred during this project, but like other measurements methods (like McCabe) we suggest a configurability scale. For example, if 0 files of the fundamental structure need to be changed, then the application is 100% configurable. If 1-10 files need to be changed, the application is 90% configurable and so on.

There is another problem which can occur. What if the separation of fundamental structure and configurable components is not clear enough? For example, in very large and complex software systems. Then it is very hard to count the files belonging to the fundamental structure and the method we propose would be useless. Therefore, this method can only be used when the fundamental structure is completely clear. Our measurement method might look different if we had chosen another definition. But, as said earlier, the right configurability definition depends on the stakeholders needs (see section 1.5). Future work must be done in order to create a general method to measure configurability for all kind of applications and architectural structures.

9.4 Discussion - Configuration or Programming

The purpose of the configuration language was to abstract the underlying complexity of the application and to avoid changing the Java source code at deployment time. In this way, most involved stakeholders (non-technical) can change the application (see Appendix A).

During the development of the configuration language, too much functionality was implemented in the language. For example, conditions to control the complete workflow (see Section 6.4.2) of the application. When such functionalities are provided, a (workflow) language to program the application is created instead of just a language to configure the application. If that was the case, the subject of this dissertation was likely "how to create a programming language for configuring web applications". In addition, a programming language introduces complexity, something we wanted to avoid in order to make the language understandable for non-technical stakeholders. Creating a programming language, just to configure the variation points, could change the whole project.

Therefore, the question raised: "do we designing a configuration language or programming language?". When is something defined as configuration and when as programming? There are many discussions going on about the answer to this question. To indicate the difficulty, some answers from random people are listed below:

- Programming is when a language is Turing complete, otherwise its configuration
- When the language as a McCabe complexity bigger than 2 (>2), then it is programming!
- Programming is when you can compile the language!
- Programming is the same as configuration, because both will give direct or indirect instruction to the microprocessor;
- Treat configuration as a first-class citizen that requires design, testing, version control, and deployment management" [EIP06]
- When you create to many abstraction in your configuration, it becomes programming instead of configuration [enterprise patterns]. Where is the edge?
- Configuration is changing key/value pairs in a configuration dialog screen, programming is for technical people.

As one can see, lots of research has to be done to define what configuration and programming is. However, there is still no clear definition of a configuration language.

The provided functionalities within the language are now based on the stakeholders feature requirements (see Appendix A) and the binding time (see Section 1.4). When requirements of the stakeholders change, the language needs to be changed as well. As said before, the grammar of an XML based language is easy to maintain and therefore a good solution to create a configuration language. However, when too much functionality is added to the language, it can become too complex and difficult to understand for non-technical stakeholders. It is a trade-off between much functionality and a complex language, or less functionality and a simple language to provide only those features stakeholders want.

We decided to provide just enough (also an ambiguous term) functionality to configure the application (based on the two variation points in the base architecture) and reduce programming conditions to a minimum, so non-technical stakeholders can easily change the application at deployment time. Where easily means; configuration of the application within one hour. This number is based on our own experiences with non-technical stakeholders, but needs to be validated in a life case. Because the language provides a minimum of programming conditions (it only supports iterations), we labelled the language as a configuration language. Nevertheless, the definition of programming and configuration is still unclear.

9.5 Future work

The configuration language is still in prototype phase. This project used two variation points upon which the language was build. Therefore, more research is necessary in order to refine the syntax to support more variation points. Automatic framework API generation of the language by using the syntax definition must be studied in order to easily update language changes. For example, to support other variation points like configurable security settings.
Future research must be done on deployment and run-time configuration. In addition, the abilities of implementing a complete SOAP protocol in RIA applications is beneficial, so it can communicate with web services. Once this isrealized, we can think about creating distributed user interface components via web services. For example, a web service which provide a tree component or a web service which provide a menu component. You can consume those web services in your RIA user interface.
Also, other methods to measure configurability must be researched to provide a wide scale of validation techniques for the configurability quality attribute. For example, measure configurability when there is no clear fundamental structure defined or measure configurability at run-time.

9.6 Conclusion

With Rich Internet Applications and a configuration language we have designed and constructed an application, which is able to adopt new requirements and operating environments without changing the fundamental structure of the web application. This was successfully validated by the proposed method (see section 2.7) to measure configurability.

Based on a configuration language together with RIA software, a configurable web application can be created with possibilities of creating a product line with variants of data sources and user interfaces. Of course, there are other solutions to this problem. But the fact is that you need RIA to increase flexibility and to create independent web user interfaces.

Because of the flexibility RIA provides, it is much easier to customize the web application. And because of the similarities with traditional desktop client application, it is easy to detach the RIA user interface and attach a desktop client technique, like a command line application.

RIA can help to create configurable web applications, with a minimum of development effort. By using a component based RIA architecture, you can reuse components to speed up prototype development. Later these components can used in the final application with a minimum of changes. By using a declarative user interface language, like Backbase BXML, you abstract the complexity of AJAX JavaScript and speed up the development process of RIA without plug-ins (like Flex or Java Applets).

Within this dissertation the following contributions were made:
- An overview of available web technologies to support configurability (Section 3.1);
- A base architecture, which supports deployment and run-time configurability (Chapter 5);
- A dashboard pilot, built out of several prototypes. Every prototype will cover an element of the defined architecture, which supports configurability. It will give insight into the possibilities and functioning of some of the techniques that were discovered during research (Chapter 8);
- A specification of the configuration language (Chapter 6);
- A method to measure configurability (Section 2.7).

From a practical point of view, the following contributions were made for the SIG organisation:
- Requirements of the dashboard web application, specific for the case (Appendix A);
- Description of the construction process for a rich user Internet application by using a user centric design approach (Section 7.2)
- A dashboard pilot application (Chapter 7 and 8) to support variant data sources and user interfaces.

10 Evaluation

In this chapter, the project results will be evaluated. First positive and negative results were evaluated. Finally, results for the employer and reflection on the research approach will be described.

10.1 Positive results

During the project much knowledge was gathered about variability, configuration (validation) techniques and product lines. Originally, lots of research was done on abstract user interface languages, but we did not use it because stakeholders changed their opinions about the function of the user interface during the project. But because of the research, new insight and knowledge was gathered about abstract languages and was useful in order to develop the configuration language.
Because of the user centric design approach, there were two general stakeholder presentations given and several individual stakeholder sessions to elicit requirements. These presentations and sessions contribute to the communicative skills. In addition, I learned a lot about writing a large document in a secondary language, which improved my English writing skills.
The development of the prototypes went well because of the development experience gained earlier, but some technical skills, like Java programming, were improved.

10.2 Negative results

The research question was changed several times during the project, because earlier research results were giving new insights to the SIG organisation. Therefore much research was irrelevant for the final approach and even during the project research was done. Several technical environment problems, slowed down the development process, but were not critical for the final results.

10.3 Results for employer

With the use of the configuration language, technical stakeholders are able to replace the user interface and data source without changing the fundamental structure of the application. They can configure the application based on the variation points indicated in the base architecture.
Non-technical stakeholders can configure the dashboard application by using the RIA client. They can define their own custom data sets and relations and customize the way the data is represented by using of-the-self table and chart components which are freely available on the internet. Because of the ability to define data sets and visualisation, no more technical stakeholders are involved during the configuration process.
The organisation learned about creating rich Internet applications and techniques for how they can be used with new and existing projects. By using rich Internet application techniques, the existing development process could be improved.

10.4 Reflection on research approach

The research approach and the techniques used in this project were very successful. Because of the user centric design approach, the stakeholders were involved during the complete development lifecycle and requirements could be improved. The feedback about the project was positive.

References

Books
[Bass03] L. Bass et al., *Software Architecture in Practice 2nd Edition*, 2003, Addison-Wesley, ISBN: 0-321-15495-9

[Beck05] K. Beck, *Extreme Programming Explained*, 2005, Addison-Wesley, ISBN: 0-321-27865-8

[Cooper03] Alan Cooper, Robert M. Reimann, About Face 2.0: The Essentials of Interaction Design, Wiley; 1st edition (March 17, 2003)
ISBN: 0764526413

[Few06] Stephen Few, Book: *Information Dashboard Design*, January 2006, O'Reilly
Pub Date: Print ISBN-10: 0-596-10016-7
Print ISBN-13: 978-0-59-610016-2

[Harold04] Elliotte Rusty Harold, W. Scott Means, *XML in a Nutshell, 3rd Edition,* September 2004, O'Reilly,
ISBN: 0-596-00764-7

[Kay04] Michael Kay, *XSLT 2.0 Programmer's Reference, 3rd Edition*, August 2004, ISBN: 0-7645-6909-0

[Lauesen02] S. Lauesen, *Software Requirements*, 2002, Addison-Wesley, ISBN:0-201-74570-4

[Snyder03] Carolyn Snyder, *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces,* Morgan Kaufmann (April 2, 2003)
ISBN: 1558608702

[Wutka00] M. Wutka, *Using Java Server Pages and Servlets Special Edition*, 2000, QUE, ISBN: 0-7897-2441-3

Papers
User interface Techniques
[Auer03] Martin Auer, Bernhard Graser, Stefan Biffl, *A Survey on the Fitness of Commercial Software Metric Tools for Service in Heterogeneous Environments: Common Pitfalls*, metrics, p. 144, Ninth International Software Metrics Symposium (METRICS'03), 2003.

[Dryer97] D. Christopher Dryer, *Wizards, Guides, and Beyond: Rational and Empirical Methods for Selecting Optimal Intelligent User Interface Agents*, 1997, Proceedings of the 2nd international conference on Intelligent user interfaces table of contents, Pages: 265 – 268, Orlando, Florida, United States , ACM Press
ISBN:0-89791-839-8

[Memon02] Atif M. Memon, *GUI Testing: Pitfalls and Process*, Computer, vol. 35, no. 8, pp. 87-88, Aug., 2002.

XIML
[Goschka01] K. Goschka, R. Smeikal, *Interaction Markup Language - An Open Interface for Device Independent Interaction with E-Commerce Applications*, January 2001, 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 7, pp. 7074

UIML
[Phanouriou01] Constantinos Phanouriou et al., *UIML: A Device-Independent User Interface Markup Language*, Apr 2001, DAI-B 61/10, p. 5412 / ACM, ISBN:0-599-98462-7

Charts
[Tsoulos03] Prof. Tsoulos L et al., *An XML-based approach for the composition of maps and charts*, August 2003, 21st International Cartographic Conference 2003 Durban, South Africa 10 - 16

RIA
[Duhl03] Duhl J., *Rich Internet Applications*, IDC white papers, 2003, http://www.idc.com

[Mesbah06] A. Mesbah ; K. Broenink ; A. van Deursen, *SPIAR: An architectural style for single page internet applications*, 2006, SEN-R0603, ISSN 1386-369X, http://db.cwi.nl/rapporten/abstract.php?abstractnr=2022

[Paulson05] Linda Dailey Paulson, *Building Rich Web Applications with Ajax*, October 2005, Computer, 2005, IEEE

[Preciado05] J.C. Preciado, *Necessity of methodologies to model Rich Internet Applications*, 2005, IEEE

Configurability
[Firesmith03] Donald G. Firesmith, *Common Concepts Underlying Safety, Security, and Survivability Engineering*, December 2003, SEI

[Losavio03] Francisca Losavio, Dinarle Ortega, Maria Perez, *Towards a Standard EAI Quality Terminology*, sccc, p. 119, XXIII International Conference of the Chilean Computer Science Society, 2003.

Variability
[Anastasopoulos01] Anastasopoulos, M., Gacek, C., 2001. Implementing product line variabilities. In: Symposium on Software Reusability (SSR'01),Toronto, Canada, Software Engineering Notes 26 (3) 109–117

[Bragança04] Alexandre Bragança and Ricardo J. Machado, *Run-Time Variability Issues in Software Product Lines*, Proceedings of the Workshop on Implementation of Software Product Lines and Reusable Components, ICSR'8, Madrid, June 2004.
http://www.dei.isep.ipp.pt/~alex/publications.htm

[Bosch04] Bosch, J., *Variability Management Software Product Families and Populations*, december 2004, In Proceedings of the 2nd Groningen Workshop on Software, SVM2004, Groningen, The Netherlands

[Svahnberg01] M. Svahnberg, J. van Gurp, J. Bosch, *A Taxonomy of Variability Realization Techniques*, June 2001.

W3C Standards
[HTTP] http://www.w3.org/Protocols/ (7/8/2006)
[HTML] http://www.w3.org/MarkUp/ (7/8/2006)
[SOAP] http://www.w3.org/TR/soap/ (7/8/2006)
[XML] http://www.w3.org/XML/ (7/8/2006)
[XSLT] http://www.w3.org/TR/xslt (7/8/2006)

Websites
[Backbase06] Official Backbase website: www.backbase.com (7/8/2006)

[Castor06] http://www.castor.org (14/7/2006)

[CodeFutures06] http://www.codefutures.com/java-dao/ (14/7/2006)

[DieNet06] http://dict.die.net (14/7/2006)

[EIP06] http://www.enterpriseintegrationpatterns.com/ramblings/30_configure.html (7/8/2006)

[FusionCharts06] www.infosoftglobal.com/FusionCharts/ (14/7/2006)

[Hibernate06] http://www.hibernate.org (14/7/2006)

[IDDesign06] http://www.interaction-design.org/ (14/7/2006)

[JAXB06] http://java.sun.com/webservices/jaxb/ (14/7/2006)

[KavaChart06] http://www.ve.com/kavachart/index.html (14/8/2006)

[Laszlo06] http://www.laszlosystems.com/ (7/8/2006)

[Macromedia06] Macromedia Flex: http://www.macromedia.com/software/flex/ , (7/8/2006)

[MSDN06] http://msdn2.microsoft.com/en-us/default.aspx (7/8/2006)

[Sommerville06] http://cordis.europa.eu/ist/ka4/tesss/newsletter1.htm, System Configurability Definition (7/8/2006)

[Spring06] http://www.springframework.org (14/7/2006)

[Sun06] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html (7/8/2006)

[WikePedia06] http://en.wikipedia.org (14/7/2006)

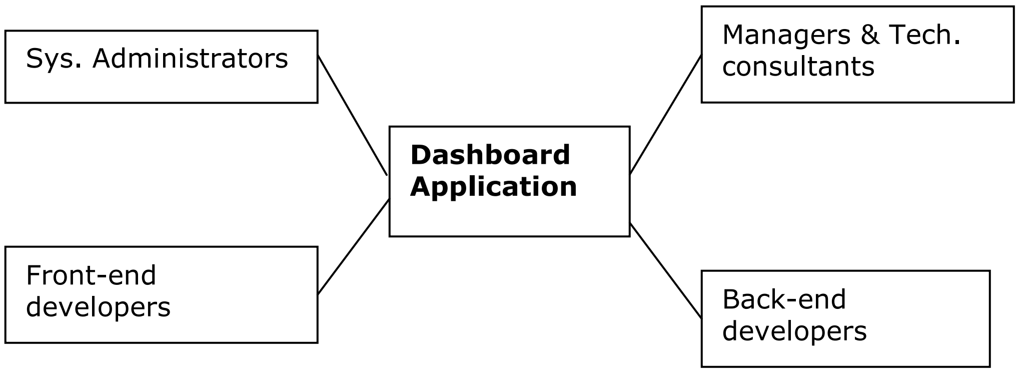
[XUL06] http://www.mozilla.org/xpfe/xptoolkit/ (6/7/2006)

Appendix A, Pilot Feature Requirements

Feature requirements are the high (or domain) level requirements of an application. This chapter describes the process to elicit the feature requirements, this dissertation did used in order to create the dashboard pilot application.

The following objectives were defined:
- Define context diagram
- Define what stakeholders are involved?
- Define what has to be configurable?
- Define which stakeholders profits?

Context diagram
To define what has to be configurable, we first define the current context of the application. Based on the stakeholders needs, we define the feature requirements. Each stakeholder want to configure the application on his own way. Afterwards, we defined the architecture. The outside worlds of this context diagram has a relation with the current dashboard application.



Stakeholders:
Stakeholders are the people who are needed to ensure the success of the project [Lauesen02], for instance the technical manager or developer. It is essential to find all the stakeholder groups and find out their interests. We define the following stakeholders: Managers, technical consultants, system administrators, back-end developers and front-end developers.

Feature Requirements
Interviews techniques were used [Lauesen02] to elicit the feature requirements of each involved stakeholder.
Features about configurability were the most imported for almost every stakeholder. So, what has to be configurable and for who? We identify two stakeholder categories:
- Technical; every stakeholder who has access to (parts of) the application 's source code.
- Non-technical; every stakeholder who does not has access to any source code of the application.

Every technical stakeholder wants to configure the web application, without affecting the part of the application from a other domain. So back-end developers do not want to change to presentation logic and front-end developers do not want to change application logic. System administrators do not want to affect both sides, but only

want to configure the environment (for example the location of the data base server).
All non-technical stakeholders only want to configure visually the graphical representation of the data. For example they want to change a pie chart into a bar chart or changing input data sets for the charts.
The main features of the dashboard pilot are:

#	Feature requirements	Stakeholder interests
R1	The application shall be easy accessible by the use of a web browser client.	Technical consultants
R2	The application shall be user interface and data source independent, so the application can be used by multiple user interfaces and data sources.	Front-end and Back-end developers
R3	The application shall be able to display stored metrics information about a software via a web browser.	Technical consultants, managers
R4	The application shall be able to hide complex data relations from the end user	Technical consultants, managers
R5	The application shall be able to let end users visually define their own data filters from a data source	Technical consultants, managers
R6	End-user are able to select the way the want to display their metric data sets by selecting a display template [Few06] en [Auer03].	Technical consultants, managers
R7	The application shall be able to display data sets (defined by the end user), so stakeholders can analyse and give advice to customers about the quality of their existing software system.	Technical consultants, managers
R8	The application must be able to fulfill the dashboard requirements according [boek]	Technical consultants, managers
R9	The application must be able to grow with future user interface technologies.	Front-end Developers
R10	The application logic must be developed in Java 1.4 language	Back-end developers
R11	The environment of the application must be configurable.	System Administrators

Appendix B, Configuration Language XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <element name="application">
      <complexType>
         <sequence>
            <element name="name" type="string"/>
            <element name="resources">
               <complexType>
                  <sequence>
                     <element name="transformation-directory">
                        <complexType>
                           <sequence/>
                           <attribute name="src" type="string"/>
                        </complexType>
                     </element>
                     <element name="schema-directory">
                        <complexType>
                           <sequence/>
                           <attribute name="src" type="string"/>
                        </complexType>
                     </element>
                     <element name="chart-directory">
                        <complexType>
                           <sequence/>
                           <attribute name="src" type="string"/>
                        </complexType>
                     </element>
                  </sequence>
               </complexType>
            </element>
            <element name="targets">
               <complexType>
                  <sequence>
                     <element name="target" maxOccurs="unbounded">
                        <complexType>
                           <sequence/>
                           <attribute name="name" type="string"/>
                        </complexType>
                     </element>
                  </sequence>
               </complexType>
            </element>
            <element name="transformers">
               <complexType>
                  <sequence>
                     <element name="transformer" maxOccurs="unbounded">
                        <complexType>
                           <sequence>
                              <element name="params">
                                 <complexType>
                                    <sequence>
                                       <element name="param">
                                          <complexType>
                                             <sequence/>
                                             <attribute name="name" type="string"/>
                                          </complexType>
                                       </element>
                                    </sequence>
                                 </complexType>
                              </element>
                           </sequence>
                           <attribute name="id" type="string"/>
                           <attribute name="stylesheet" type="string"/>
                        </complexType>
                     </element>
                  </sequence>
               </complexType>
            </element>
            <element name="datasources">
               <complexType>
                  <sequence>
                     <element name="datasource" maxOccurs="unbounded">
                        <complexType>
                           <sequence>
                              <element name="database">
                                 <complexType>
                                    <sequence/>
                                    <attribute name="name" type="string"/>
                                    <attribute name="user" type="string"/>
                                    <attribute name="password" type="string"/>
                                    <attribute name="server" type="string"/>
                                 </complexType>
                              </element>
                              <element name="filters">
                                 <complexType>
                                    <sequence>
                                       <element name="filter" maxOccurs="unbounded">
                                          <complexType>
                                             <sequence>
                                                <element name="hibernate-query"
                                                type="string"/>
                                                <element name="output-format"
                                                xmlns="http://www.sig.nl/sig" >
                                                      CONTINUES AS XSD FORMAT IN THIS
                                                      NAMESPACE
                                                </element>
                                                <element name="params">
                                                   <complexType>
                                                      <sequence>
                                                         <element name="param"
                                                         minOccurs="0"
                                                         maxOccurs="unbounded">
                                                            <complexType>
                                                               <sequence/>
                                                                  <attribute name="name"
                                                                  type="string"/>
                                                               <attribute name="type"
                                                               type="string"/>
                                                            </complexType>
                                                         </element>
                                                      </sequence>
                                                   </complexType>
                                                </element>
                                                <element name="dependencies">
                                                   <complexType>
                                                      <sequence>
                                                         <element name="dependency"
                                                         minOccurs="0"
                                                         maxOccurs="unbounded">
                                                            <complexType>
                                                               <sequence>
                                                                  <element name="mapping">
                                                                     <complexType>
                                                                        <sequence/>
                                                                           <attribute
                                                                           name="map-
                                                                           element"
                                                                           type="string"/
                                                                           >
                                                                        <attribute
                                                                        name="map-to-
                                                                        param"
                                                                        type="string"/>
                                                                     </complexType>
                                                                  </element>
                                                               </sequence>
```

```xml
                                                               <attribute name="filterID"
                                                               type="string"/>
                                                               <attribute name="filter"
                                                               type="string"/>
                                                            </complexType>
                                                         </element>
                                                      </sequence>
                                                   </complexType>
                                                </element>
                                             </sequence>
                                             <attribute name="id" type="string"/>
                                             <attribute name="name" type="string"/>
                                          </complexType>
                                       </element>
                                    </sequence>
                                 </complexType>
                              </element>
                           </sequence>
                           <attribute name="name" type="string"/>
                           <attribute name="default" type="boolean"/>
                           <attribute name="type" type="string"/>
                           <attribute name="adapter" type="string"/>
                        </complexType>
                     </element>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
</schema>
```