CHALMERS

Refactoring of a GUI prototype using AJAX

Improve the performance of a GUI prototype by introducing an AJAX framework.

Master of Science Thesis

EMANUELLA WALLIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, May 2009

Refactoring of a GUI prototype using AJAX
Improve the performance of a GUI prototype by introducing an AJAX framework.

Emanuella Wallin

Examiner: Sven-Arne Andreasson

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2009

Contents

Refactor a GUI prototype using AJAX
Improve the performance of a GUI prototype by introducing an AJAX framework.
Emanuella Wallin
Department of Information Technology
Chalmers University of Technology

Abstract

The problem given from the company XDIN was to improve an existing Java EE web application prototype which handles errands specified for the energy branch. Since the prototype had a slow work process and a slow response time, the company wants the new system to be implemented with AJAX and DWR. Furthermore, the task was also to examine if it is beneficial to use other AJAX frameworks, in excess of DWR. Since there are many different frameworks the review was limited to include two of the frameworks that are in the frontline, DOJO and YUI.

A lot of different technologies were used, such as JSP, JS, HTML, AJAX, DOJO, YUI, EJB, Struts, SQL and CSS (Cascading Style Sheet). Therefore the first step was to learn all the different technologies by reading tutorials and API's and attending workshops. The second step was to understand the requirements of the system. Development of the code was done using both Eclipse and IntelliJ, and for testing the UI, a tool called firebug was used.

DWR is a framework that advocates asynchronous communication and makes it possible to do lightweight server roundtrips without any complete page refresh. By the very nature of the framework, the response times were improved.

All three JavaScript libraries have pros and cons. DOJO has a lot of features and gives the impact of being a library to trust because of its diversity. However, since there is no documentation or complete API it makes it really difficult and cumbersome to use. The key feature with a JavaScript library should be ease of use, and by that also a time saver to develop with when compared to other frameworks or no framework at all. DOJO lacks in these areas, and is not the framework of choice largely because of these. The YUI on the other hand not only has a lot of features, but comes with a comprehensive API complemented with examples and cheat sheets. As a result of the drawbacks of DOJO the choice was to use YUI as a primary JavaScript Library and to use YUI ext as an extension.

Keywords: AJAX, DWR, YUI, DOJO, YUI ext, RIA.

1. Introduction

The problem given from the company XDIN is to improve an existing Java EE web application prototype which handles errands specified for the energy branch. Since the prototype has a slow work process and a slow response time, the company wants the new system to be implemented with AJAX (Asynchronous JavaScript and XML - eXtensible Markup Language) and DWR (Direct Web Remoting). New AJAX Frameworks are under development and therefore the task is also to choose one type of framework and examine its usability by using the chosen framework in the implementation of the system.

2. Background

The motive to the task is to improve an existing prototype, where an administrator can customize the different kinds of errands for the energy branch, i.e. give them certain properties. A property can for example be what kind of timestamps should be used in relation to the errand. The existing prototype is very slow and built in a way where each user interaction results in a full page load and is therefore not user-friendly. In addition to improve the prototype the task is also to examine AJAX and one AJAX Framework for future development in the company.

Users using conventional web application cannot perform anything while waiting for a response. Loss of information and the scrolled position of the screen is also a big problem and these are some reasons why Rich Internet Applications (RIA) technologies were developed. The same reasons apply to why the company wished for a new Errandhandler/CaseBuilder. AJAX is a RIA technology and is the technology that will be used to develop the new Errandhandler.

3. The Company

Xdin is one of Sweden's fastest growing technical consultancy companies developing products and processes mainly for clients from the vehicle, energy and manufacturing industries.

It has over 500 employees with offices in Sweden and the US.

Xdin's consultants develop world-leading products and processes for innovative companies and organizations by providing services, training and creative tools.

The concern is founded on developing their clients’ products and processes which are made possible by their competent employees – Xdin's single most important resource. They focus on delivering the right competence, at the right price at the right time - Develop and Deliver!

The main areas of business are:

- Product Development
- IT- and System Development

Training, Support and Method Development

Xdin – a growth company. Xdin's average growth since its creation in 1991 has been around 60% per annum. The same figure for the last three years has been around 18% per annum.

Xdin enjoys a high ranking. The highly respected Swedish newspaper Affärsvärlden, that conducts an annual ranking of Sweden's 1500 foremost consultancy companies, amongst which Xdin was in 2005 ranked as number sixteen in the technical consultancy table.

4. Problem description

The task is to improve the response time of an existing Java EE web application prototype (also called Casebuild) which handles errands. The system handles the errands mainly by the pattern CRUD (Create Read Update and Delete). The system is a part of a larger system, WMS (Workflow Management System), which already has the underlying architecture necessary to provide a better, faster and a more user friendly web application, the bottleneck is the user interface of the system. WMS is a system specially built for the energy branch and provides the necessary workflow according the processes needed in the energy branch, ex. the move process. Users using conventional web application cannot perform anything while waiting for a response. Rich Internet Applications (RIA) technologies have been developed to deal with these common types of problems. The technologies AJAX and DWR which are RIA technologies will be used to improve the user interface.

DWR and AJAX are not new on the market however a lot of different AJAX frameworks are under development and therefore the task is furthermore to find out why developers should use these kinds of frameworks. Since, there are several frameworks on the market, many developers, including me, are eager to know which framework to use.

The prototype is written with JSP (Java Server Pages), JS (Java Script) and HTML (HyperText Markup Language).

The underlying architecture is built with EJB (Enterprise JavaBeans), Struts, Java and SQL (Structured Query Language). Even though there is a fully developed underlying architecture some rewriting will of course be necessary.

The information given was the code of the prototype and WMS, an ER (Entity Relation) diagram over the database and a presentation of the whole system called WMS. Furthermore, a primitive requirement specification was given.

Additionally, the WMS requires at least IE6 (Internet Explorer) to run properly, and this will be the requirement for the new application as well.

So to summarize the task is to;

- Improve the response time for the end users in an existing Java EE web application, using DWR.
- Examine if it is beneficial to use other AJAX frameworks, in excess of DWR?
- Review some of the AJAX frameworks that are in the frontline and use one of them in the application.

Limitations

Since there are many different frameworks I have limited the review to include two of the frameworks that are in the frontline, DOJO and YUI.

Method

A lot of different technologies will be used, such as JSP, JS, HTML, AJAX, DOJO, YUI, EJB, Struts, SQL and CSS (Cascading Style Sheet). Therefore the first step is to learn all the different technologies by reading tutorials and API's and attending workshops. The second step will be to understand the requirements of the system. After this, next step will be to implement a user interface prototype, showing only dummy data. When the prototype is approved by the company the user interface should be connected with the server code. Changes or additions to the server side code may be necessary.

Development of the code will be made using both Eclipse and IntelliJ, and for testing the UI, a tool called firebug is available as a plugin to the firefox web browser.

5. Analysis

5.1. Asynchronous JavaScript and XML AJAX

Ajax is a web development technique for creating interactive web applications. Ajax was born because of the lack of RIA in conventional web applications. Users using conventional web application cannot perform anything while waiting for a response. Ajax allows a more responsive web page, by exchanging small amounts of data with the server so that the complete web page does not have to be reloaded on every interaction by the user, i.e. Ajax loads on demand. What happens is actually that HTML is locally generated within the browser and only JavaScript calls and the data is brought down. As the payload is much smaller the web pages appear to load faster.

Java is synchronous but Ajax is asynchronous and does not interfere with normal page loading. The existing standards that are used are many but the programmer can choose to use any language that works for him. The Ajax approach helps the programmer to clearly separate methods and formats.

The standards that are most used are:

- *JavaScript* is used to write Ajax function call.
- *XML* is used to format the retrieved data. It is not necessary to use XML other formats work also like plain text and JSON.
- *CSS* is used for formatting the style; of course XHTML and HTML can be used instead if CSS is not preferred.
- *DOM (Document Object Model)* is a model representing HTML or XML and related formats, i.e. the way that the JS sees its containing HTML page.
- *XMLHttpRequest* is the core of Ajax.

Since Ajax is based on open standards it can be used on different operating systems, computer architectures and web browsers and hence it is a cross-platform technique.

Ajax is built on the XMLHttpRequest object which makes it possible to send HTTP asynchronous calls from a web page to a web server with the help of JavaScript without having to reload a page.
The benefits are many with Ajax; however there are some disadvantages also. One is that the pages are dynamically created and can therefore not be registered in the browser history engine. The consequence of this is that the back function doesn't work as it is thought to work. A related consequence is also that the user cannot bookmark a certain state of the application.

A greater problem is that which JavaScript causes or rather the different web browsers because JavaScript is implemented differently by different browsers. JavaScript code must many times be written twice, since there are IE and Mozilla incompatibilities. However, many of the JavaScript libraries handle this problem by abstracting differences from web developer.

Yet another problem is that it is difficult to test and find bugs in JavaScript, but IDE tools like firebug and IE Developer Toolbar exists which can help the programmer a lot during development.

Implementing an AJAX oriented web application is often a very complex task and making use of an AJAX framework can greatly simplify the work. One important aspect of an AJAX framework is to abstract away the technical details of making asynchronous http requests. Not only is this important so you don't get lost in the details, but it's also important to get a common way of implementing code that runs the same on different web browsers. Most AJAX frameworks today accomplish this.

Often, such frameworks not only provide the means to make AJAX requests but also ways of coding the actual web pages in more dynamic and efficient ways. Examples of this are rich components; such as tabbed panels, calendar pickers and tree components and on a lower level there may be complete event systems and functionality for mapping data to and from input fields and tables. A framework can also serve as a great resource for ideas and know-how by providing examples and documentation for the framework in particular but also about AJAX, JavaScript and web applications in general.

5.2. Direct Web Remoting – DWR

One of the prerequisites given by the company was to use the DWR framework, since that is already used in other parts of the WMS. DWR is a Java and JavaScript library which allows you to write highly interactive web applications and it is designed specifically with Java in mind. Basically, DWR allows code in a browser to use Java methods running on a web server just as if they were in the browser.

Previous techniques for developing web applications often enforced a synchronous approach in which entire page reloads were required to make interactions with the server. DWR and AJAX in general, allows for a more dynamic approach where incremental changes can be made to a web page without requiring a full page reload.

DWR consists of two main parts, a Java Servlet, called DWRServlet, running on the server-side and some JavaScript libraries running on the client-side. The DWRServlet handles requests and responses made from the client and also exposes selected java classes so calls can be made to them from the client. JavaScript libraries that the client should use are dynamically exposed by the DWRServlet. There is also a debugger page that allows you to test the interaction with the DWRServlet without having to write any client JavaScript at all.

The client-side imports the exposed JavaScript libraries, and using these, the client code can directly make asynchronous calls to the exposed server code. When the server has finished processing such a method call, the response is serialised and returned to the client were the JavaScript callback function is invoked to further process the response. This is very much like an RPC (remote procedure call) mechanism such as SOAP (Simple Object Access Protocol), but without requiring something that's not already available in the web-browser.

For handling complex return types such as basically any kind of java object, there are a number of different converters that can be used. A converter serialises/deserialises a java class and by configuring an appropriate converter for each class, any complex object graph can be seamlessly transferred to the client in the desired way.

When using DWR to asynchronously call a method on the server, you provide DWR with a callback function that should be called when DWR has received the response from the server, see Figure 1 for an example.

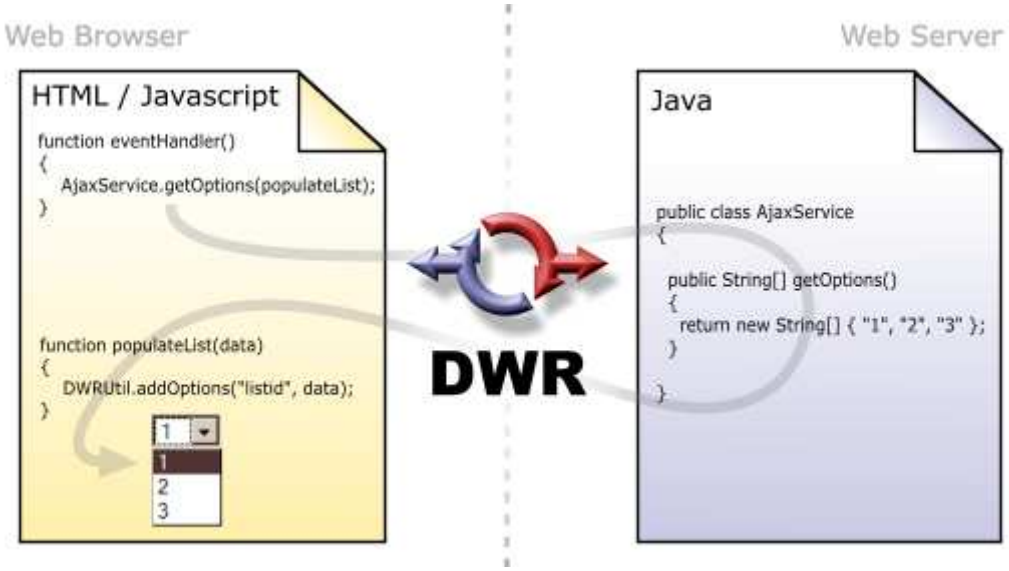


Figure 1: In this example, when the eventHandler() function is called, an asynchronous http request is made to the server where the method AjaxService.getOptions() is invoked. When the server returns the response, the callback method populateList(data) is called and the response data is supplied as an input parameter. The populateList(data) function dynamically adds a number of options to a drop down list on the web page.

DWR is a very competent framework when it comes to communicating data objects between the client and the server, and it also manages the population of these data objects into plain web page components very well. However, there are many aspects, in particular regarding the user interaction on the client side, which DWR does not try to solve.

5.3. JavaScript libraries

As the previous section identified that the DWR framework did not fully support all the required functionality, this section will compare three JavaScript frameworks that do, and choose which one to use as a complement to DWR in the development.

The comparison will emphasize general usability, support, documentation, licenses, and of course the functionality necessary for the application. The functionalities required are mainly:

- a tab component
- a tree component
- a popup window

5.3.1. Yahoo User Interface - YUI

The Yahoo User Interface (YUI) is an open source JavaScript library that makes it easier to build dynamic web applications. The YUI is released under a BSD license and free for all uses. It was developed by Yahoo's engineering team in the US.

The YUI provides both reusable utilities and controls that are very well documented. There is a fully developed API, cheat sheets and a lot of different examples to follow, which makes it easy to start using. The YUI also provides a blog, maintained by its developers, where the latest information is shared and lots of useful discussions and examples are emphasized.

There are many utilities in the YUI that makes it easier to code e.g. events, drag and drop, animation, etc. In addition to this, there are controls, or components, like e.g. menus, calendars, and sliders as well as all components required for the application. It is important to note that adopting the YUI doesn't imply that you have to use everything that the YUI provides, nor does it prevent the usage of other JavaScript libraries.

There is a tab component called TabView that makes it easy to create tabbed layouts in a number of different ways; dynamically from the code, from marked up div containers, or from external files. The option to create tabs from external files gives a good separation of concerns, but might cause issues with JavaScript calls prior to page initialisation. By making sure that the YUI gets a chance to initialise the tabs before executing application specific initialisations, there should be no problem.

A tree component is implemented in the TreeView control. The TreeView control supports dynamic loading of data when nodes are expanded, but not native rearrangement of nodes (as is required in some of the trees)

Popup windows are easily implemented by using the Dialog control, which allows any DIV container to be rendered as a popup window. The Dialog also allows custom callback functions to be specified on the dialog buttons. A drawback of using the Dialog is that the buttons cannot be assigned an individual CSS class, since they are grouped together. Any CSS class is thus applied to all buttons.

The YUI framework is divided into modules represented by separate JavaScript files. This makes it easy to just include the functionality required on the html page, which in turn improves download performance.

Integration to the server-side code is possible by using the ConnectionManager provided in the YUI, or by using a DataSource abstraction that can retrieve data from a remote service e.g. by using Ajax. However, the integration in this case is best done using DWR instead, since that's a prerequisite for this project. Using DWR is entirely possible, but it can put a limitation on the usability of some heavy components that rely on e.g. the ConnectionManager.

It's easy to change the default CSS style of the YUI components by overriding the setting for relevant CSS classes. However, in a case like this where common CSS guidelines are provided by the company, one need to duplicate this style to adapt the YUI components. This means that changes in the common CSS will need to propagate to the YUI specific CSS.

The YUI doesn't promise to support every browser; however it does support IE 6.0 and IE 7.0, which are the important ones in this context. There is also a promise that all components work well in all of what is called "A-Grade" browsers, see figure 2.

	Win 98	Win 2000	Win XP	Win Vista	Mac 10.3.x	Mac 10.4
IE 7.0			A-grade	A-grade		
IE 6.0	A-grade	A-grade	A-grade			
Firefox 2.†	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade
Firefox 1.5.†	A-grade	A-grade	A-grade		A-grade	A-grade
Opera 9.†	A-grade	A-grade	A-grade		A-grade	A-grade
Safari 2.0†						A-grade

Figure 2: Browser compatibility chart.

5.3.2. *YUI EXT – Ext*

The YUI Ext started by Jack Slocum in early 2006 and was supposed to be a set of extension utilities for the YUI. The YUI Ext grew in popularity and became an independent library. In the early 2007 a company was formed and a new name was taken, Ext, thus it is now an independent client-side JavaScript framework for building web applications and is dual-licensed under the LGPL and a commercial license.

Ext is compatible with most server platforms that can process POST requests and return structured data. DWR is a common choice for integration with Java Projects.

Ext can now be used alone since it includes a native Ext adapter. Before version 1.1 it was necessary to use one of the following external base libraries: the YUI, jQuery or Prototype/Script.aculo.us. Ext provides both tutorials, interactive demos, a fully developed API, community manual and a blog.

A feature in the Ext that stands out from both the YUI and DOJO is the possibility to use templates when building dynamic html within the JavaScript code. This feature is both fast

performing as it doesn't require string concatenation and the code gets cleaner because the templates contain pure html which is easy to read.

```
function relationCallback(relations) {

  // define the html template
  var html = '<div><span class="relationsErrandName">{name}</span>' +
             '<span class="relationsRelationName">{relation}</span></div>';

  // compile the template
  var tpl = new YAHOO.ext.DomHelper.Template(html);
  tpl.compile();

  // Instantiate the template for each 'Relation'
  // and add it to the html div with id 'relations'
  for (var i=0; i<relations.length; i++) {
    tpl.append('relations', {
      name: relations[i].errand.name,
      relation: relations[i].relation
    });
  }
}
```

Figure 3: Example of how to use the template functionality in YUI Ext to build dynamic html efficiently.

5.3.3. *DOJO*

DOJO is an open source JavaScript library or rather a toolkit which makes it easy to build dynamic web applications and is released under a BSD license and free for all uses. DOJO provides a lot of different components that can be used to make web pages more useable, responsive and functional. Dojo doesn't call their outcome a library but a toolkit since it's more than just a library. However, Dojo still provides a lot of libraries but these are wrapped in a package system which makes it more than just a collection of libraries. This package system ensures that only code that is required will be included. The usage of DOJO in an application does not prevent the possibility to use another JavaScript library in the same application.

A benefit with Dojo is that the programmer can choose to include as little or as much of the available API's that is needed. The API contains units called modules that resemble packages in Java except that modules can also include constructors and methods. A module is often defined in a single JavaScript file, but can sometimes be divided into several files to enhance the performance, since then the browser only needs to download the code that is necessary.

Dojo has a concept called Widgets, which is a functionality that enhances existing html components making them e.g. more user-friendly or adds more functionality such as searchable select boxes or input validation.

Dojo provides at the time no documentation or API description. However, they do provide primitive code examples describing how to use some of the widgets.

Support is available for most of the existing modern browsers today, such as Safari, Opera, IE, Firefox and Konqueror.

There is a widget called TabContainer which can be used to get a tabbed user interface. TabContainer primarily creates tabs from marked up div containers. Furthermore, Dojo provides widgets for creating both a dialog and a tree, which works similarly like the TabContainer. The TreeWidget can also be created programmatically. It is important to note that the Dialog can only be placed in the centre of the window.

The CSS can be customised in a similar way as for YUI, i.e. by providing a separate CSS file overriding the default setting for the relevant CSS-classes you whish to change. The drawback is still that the company wide CSS standard will need to be duplicated and adapted for Dojo.

Figure 4: Overview of Dojo's package system

5.4. Framework comparison

This chapter will compare the three different frameworks in order to choose which framework to use for the development. The comparison will emphasize general usability, support, documentation, and of course the parts from the framework that are necessary for the application. These parts are mainly:

- a tab component
- a tree component
- a popup window

5.4.1. *Short overview comparison*

Here follows a short comparison over what the three JavaScript libraries provides and does not provides.

	Dojo	YUI	EXT
Manual	Yes*	No	Yes*
API	Yes*	Yes	Yes
Examples	Yes*	Yes	Yes
License	BSD**	BSD**	LGPL***
Blog	Yes	Yes	Yes
Button component	Yes	Yes	Yes
Tree component	Yes	Yes	Yes
Tab component	Yes	Yes	Yes
Full package needed	No	No	No

Figure 5: Feature comparison of Dojo, YUI and EXT.

* At first 'No', but as of 2008 it is 'Yes'
** The BSD License allows proprietary commercial use, and for the software released under the license to be incorporated into proprietary commercial products. Works based on the material may even be released under a proprietary license (but still must maintain the license requirements).
*** Everyone is permitted to copy and distribute verbatim copies of this license document but changing it is not allowed.

5.4.2. *Comparison*

All three JavaScript libraries provide almost the same functionality and the usage of the components works in a similar way. There are some subtle differences between the frameworks, such as e.g. the Dialog can only be positioned in the centre of the screen in Dojo, but in YUI it can be arbitrarily positioned as well as dragged by the user. But such differences are not of main importance. Another difference between them is the quality of the documentation.

At first, the DOJO toolkit was chosen as it seemed to be the most popular and widely used framework. The popularity of DOJO ensures that many people find it useful and a framework is also likely to be further developed and supported in an active community. However, the inadequate documentation of DOJO not only made it quite difficult to understand and to get

started with, but it also made it harder to see the limitations and the benefits of DOJO. A feature of DOJO is that one should only need to include the parts of DOJO that is being used and leave out the rest, but figuring out what to include and what to leave out is really difficult when the documentation doesn't state dependencies explicitly. This is a problem as it either takes time to figure out which dependencies to include or the footprint of the application grows too big if you include too much.

There were also some technical problems with getting the TabWidget in DOJO to resemble the look of the WMS application, which would be a problem since CaseBuild will be included in the WMS.

YUI on the other hand has comprehensive documentation with many examples. It also turned out to be easier to start using YUI with only a subset of its functionality than it was to get DOJO up and running with a similar subset.

As a result of these problems, the decision was made to start using YUI instead, supported with the template functionality from YUI Ext.

6. Result

The result will be explained by guiding you through the resulting application, screenshots will be provided to enhance the comprehension.

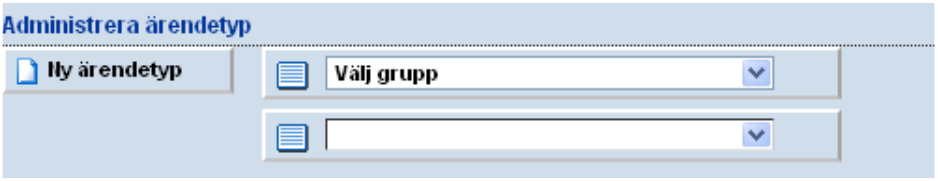


Figure 6: This is what the user first will see.

To start administrating an errand the user must first select an errand, by first choosing group and then the errand, of course the possibility to choose all groups exists. When a specific group is selected a filter method in the service class called *CaseBuildService* is called. The filter method returns a list of errands as a callback to populate the second select box. When, the user selects one specific errand the server fetches the data requested from the database. The data that is marshalled to the client is actually a composite object containing all data that will be shown in all tabs.

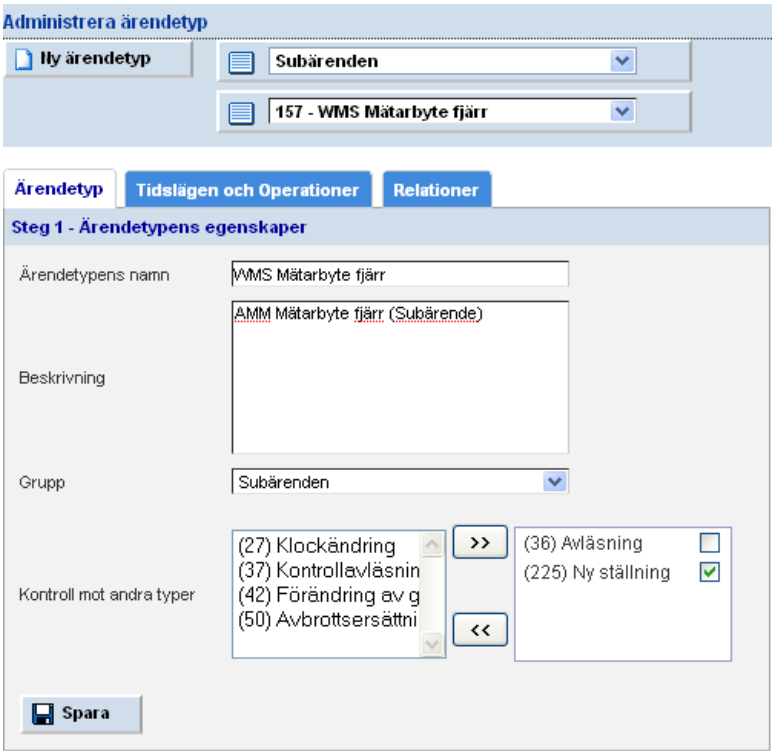


Figure 7: A case is selected and the first tab populated.

When the user has chosen an errand, two different tabs are displayed, *Ärendetyp* and *Tidslägen och Operationer*. The first tab shows the characteristics of the particular errand e.g. the name, the description, the group belonging and a control to other errands.
One requirement from the company was that every change from the user should be saved in the database along with an audit comment. In this first tab the save button actually only saves the name, the description and the group. When the user hits the save button another dialog appears and the user is requested to write a comment. If the user whish to make a control against other errands he can use the control feature, by selecting an errand and click on the button to move the errand to the right. When pushing the button the user must write a comment and the new state will be saved. The possibility to select more than one errand exists. If the user wishes to have a date control on the selected errands he can mark a radio button. When a radio button is marked a dialog appears and the users must fill in a comment. When the user no longer wants a control to other errands he can select one or more errands in the right select box and click on the button to move the errand to the left. And of course a dialog will appear and the user will be requested to write a comment. After entering a comment, a roundtrip to the server is made in the background to save the changed data. The GUI maintains its state so no page refresh is necessary.

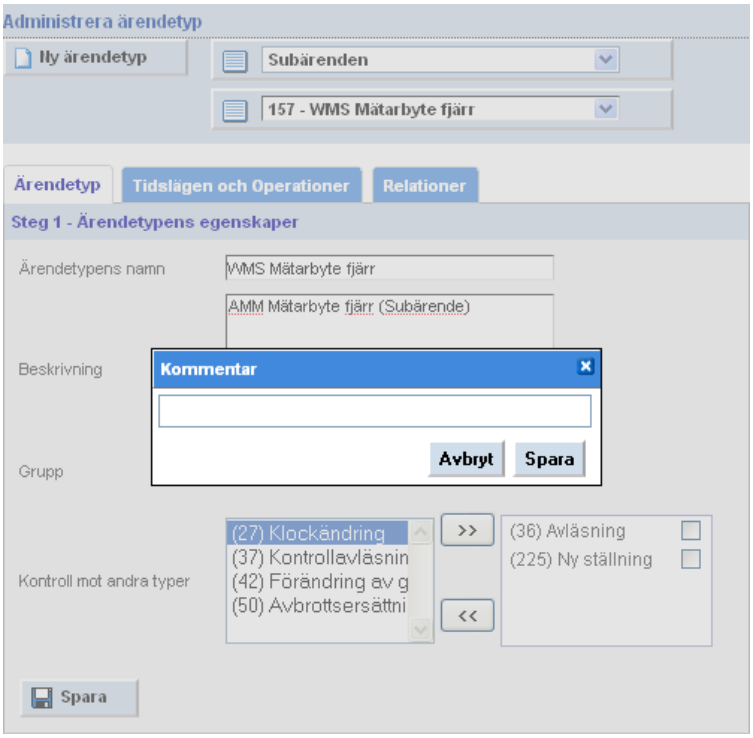


Figure 8: The comment dialog box displayed.

The background is rendered while comment dialog is visible.

To create a new errand the user simply presses the button “Ny Ärendetyp”. When the user has pressed the button the first tab will be activated and the heading of the second tab will be disabled but visible. The new errand can not be saved before the user has filled the mandatory field’s i.e. name, description and group.

Figure 9: The second tab displayed containing a tree with operations and timestamp for the errand selected.

The first tree that is visible is a static tree showing methods that are attached to the errand. The second tree is showing the activities that are attached to the errand. The order of the items in the tree is the order in which the activities will be executed in. The items are easily rearranged by clicking the up and down arrows. For each click a comment dialog will be shown and the user will be requested to write a comment. Each activity has a number of operations which are displayed as children to the parent activity. By pressing the addition sign on the left side of the tree the user can expand the tree, or by pressing the subtraction sign the tree will collapse.

By pressing the big blank document in the right corner the user can add a new activity to the tree. When the document is pressed a dialog window will be prompted were the user can select the activity to add and also set some properties on the activity. When the user presses the save button, a comment dialog will be displayed. By pressing the smaller blank document the user can add a new operation to the activity. A new activity and operation will be added on the end, and can then be moved using the arrows.

To view the properties for both the activities and the operation the user simply clicks on the name and dialog with the properties will be displayed.

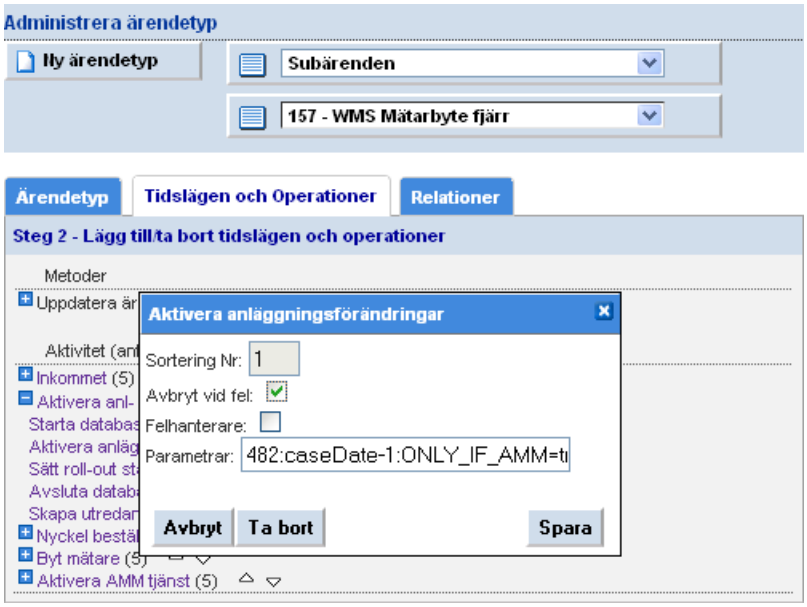


Figure 10: The dialog with the properties for an operation is displayed.

7. Discussion

Let's start by looking into what happens when the user first loads the application. What the user sees is two select boxes and a button "Ny ärendetyp". The item list that is displayed in the first select box is actually fetched from a utility called *ListBoxCacheManager,* which is loaded each time the user enters WMS. That means that no round trip to the database is necessary for this which results in better performance. This approach works fine since the group list can be seen as static because new groups are rarely added, so the chance that a new group has been added since the ListBoxCacheManager was initialised is very low. Should any list in the ListBoxCacheManager be changed anyway, it is refreshed in order to always stay current. The same approach is made when the user selects the "all groups" choice. That is all errands are fetched from the *ListBoxCacheManager,* this results in better performance since the user don't have to wait for an exhaustive search in the database. However, when a specific group has been selected a filter method is called and a list of errand items is returned to the client. The population of the list box is performed by DWR, or more specifically the DWRUtil.addOptions utility which very easily can take the result from the service callback and put it in a select box.

Apart from displaying the correct errand list when a group has been selected, the initialization of the page is done. Initialising in the sense that the dialogs and the tabs are created, but they are not displayed yet. Because of the way the browser works, no initialisation can be done before all the libraries and the entire page are loaded, and as a result a forced initialisation is done after the page has loaded properly. A consequence of this is that the first list box has to fetch the list to display from the *ListBoxCacheManager* upon page generation, since neither DWR nor YUI can be used. A positive outcome is that this approach resulted in better performance as no further roundtrips to the server are required once the page is generated because the page already contains all the initial data.

When an errand has been selected an entire composite object is fetched from the database. The composite object contains all data of an errand in a complex object tree containing many properties and different objects. This approach was a requirement from the company, as it was regarded as giving better performance than making smaller but more server requests. To fetch the whole composite object at the beginning have the positive effect that everything is actually loaded from the beginning so when the user want to switch tab the data is already present and a roundtrip to the server is not necessary. However, this approach also has some negative effects.

One of the consequences is that the user has to wait a little longer because the amount of data fetched from the database is superior. Because the UI is now done in an asynchronous approach the UI is actually loaded before the data is presented and as an outcome it is necessary to blur the other tabs while the data is fetched. If the user would have the possibility to switch to another tab before all the data is fetched and loaded there would be nothing to display.

Another consequence is that in order to prevent unnecessary reloads, the state of the composite object must be maintained in JavaScript whenever changes are posted back to the server. This design imposes an unnecessary level of complexity on the client as compared to having small but fast asynchronous server roundtrips to fetch or store data in the database. It also prevents the usage of some of the nice Ajax features of YUI/Dojo as this approach differs from the main usage pattern of the frameworks, i.e. having fast and small server interactions.

The filtered select box where you select groups is implemented with such small server interactions, and the code for that is quite small and easy to understand.

The requirement of eagerly populating all data for a selected errand was not ideal for several reasons. From a performance perspective, it adds to the start-up time before an errand can be displayed because the database must retrieve data from many tables and also the amount of data to transfer to the client can be fairly large. Not all this data is required to display the errand on the first tab so fetching the data when needed would shorten the start-up time. The design also adds a lot of complexity to the client as the entire errand must be maintained in the JavaScript code while it is selected in the GUI. There is also the risk that the errand loaded at one client gets outdated when the same errand is being worked on from another client. Furthermore, this approach deviates somewhat from the Ajax-way of doing things in multiple small server interactions. Granted, the save operations are performed using small server interactions in the background but the large initialisation is not. If it was possible to reconsider this requirement, it is likely that the large initialisation would be broken down into smaller operations e.g. triggered by changes in the active tab. The large benefit from doing that would be to simplify the GUI code.

Requiring the user to enter a comment after each update was a requirement for audit reasons. However, in cases e.g. when changing the position of an operation within an activity multiple times on the second tab, it can easily get annoying to enter a comment for each change. A better solution might have been to have a save button that stores all changes made to the arrangement of operations using one single comment.

The different parts of the UI are very well separated into different files, which make it quite easy to maintain and further develop the application. For example, the html structure such as the different dialogs and tabs each have their own JSP file that gets assembled into one main application html. The approach used to assemble the files is the <jsp:include> tag. This was chosen in favour of the static include-directive since each JSP file can have some dynamic content upon page load, e.g. contents of select boxes. For each JSP file there is a JavaScript file that defines the behaviour of the html components in it. Finally, there are a number of CSS files that defines the visual properties of the components. A consequence of using components from YUI, such as e.g. the popup Dialog, is that in order to have e.g. all buttons look the same there must be CSS styles defined for both regular html buttons and for the CSS classes that the YUI buttons have as default. This duplication of code is not ideal, but it is to be expected when you make use of a JavaScript library such as YUI, or Dojo for that matter, and mix it with ordinary html components.

However, sometimes the duplication is not the biggest problem, but the lack of possibility to assign e.g. a button a CSS style. In order to give each button in a button group a custom CSS class, a code modification of YUI had to be done (see appendix D). This code modification has been placed on SourceForge as a feature request for YUI, since it can be useful for other users as well, but also since modifying libraries does not help maintenance in the long run.

Even though YUI is very capable and has many features, it has its limitations, and some of the functionalities required couldn't be implemented with only the YUI library. One is the functionality that handles the control against other errands on the first tab. But this is mainly due to that the GUI and the data that is fetched and presented are so different from each other, that the widget from YUI cannot handle it. And also because the requirement was that each errand that is visible in the right should have a check box.

The other functionality that YUI couldn’t handle was actually the tree and this because the tree widget wasn’t built in the sense that there were parents with children. But in some odd way which resulted in that everything was saved in an array where only the parent knew its children but the children didn’t know its parent. However, parts of YUI could be used when implanting the tree. The new release of YUI is changed and now the children know about their parents, so the new widget could maybe used now.

8. Conclusions

So to summarize the task is to;

- Improve the response time for the end users in an existing Java EE web application, using DWR.
- Examine if it is beneficial to use other AJAX frameworks, in excess of DWR?
- Review some of the AJAX frameworks that are in the frontline and use one of them in the application.

The task was to improve the response times for the end users in the WMS application by using DWR. DWR is a framework that advocates asynchronous communication and makes it possible to do lightweight server roundtrips without any complete page refresh. By the very nature of the framework, the response times were improved.

All three JavaScript libraries have pros and cons. DOJO has a lot of features and gives the impact of being a library to trust because of its diversity. However, since there is no documentation or complete API it makes it really difficult and cumbersome to use. The key feature with a JavaScript library should be ease of use, and by that also a time saver to develop with when compared to other frameworks or no framework at all. DOJO lacks in these areas, and is not the framework of choice largely because of these. The YUI on the other hand not only has a lot of features, but comes with a comprehensive API complemented with examples and cheat sheets. The Ext library was at first not as comprehensive as the other two as it builds upon the YUI framework. However, it adds a key feature, namely a template engine, which makes it motivated to use this framework together with the YUI framework.

Both frameworks also support abstraction from browser specific details so developers can focus on the application instead of issues with different browsers.

It is difficult to get it exactly the way you want when using javascript frameworks, however it is very timesaving when you have learned it completely. The time it takes to learn a framework is well invested as you get a lot of functionality for free.

A drawback of using frameworks is that you often need to include more functionality than you desire, which can lead to heavier pages and longer download times. To some extent this reduces the motivation behind using a framework, i.e. to make a web application faster, but the great functionality you get is very often worth this extra weight compared to if you were to write all code by yourself. Of course it depends on which framework you use, and how you use it, but many come with functionality to customize the packaging so only the necessary parts are downloaded.

YUI e.g. has a light version in which all unnecessary code and comments have been removed.

It’s important to remember that using either of these JavaScript libraries fully means that the application has to be built in ways compatible with the library. When using only some parts from a library, it may be difficult at times to adapt it in the way you want.

Note: At the same time as this evaluation, DOJO was chosen to be used in a smaller project in the company. The outcome was the same, no documentation to lean on, time consuming and works fine only when development is done in restricted areas and general workflows.

9. Reference

Wellman, Dan (2008). *Learning the Yahoo! User Interface Library*. Birmingham: Packt Publishing Ltd.

Howitt, Adam (2006). *How to Design a large AJAX Application.* Duo Consulting

McCarthy, Philip (2005). *Ajax for Java developers: Ajax with Direct Web Remoting.* IBM

Cloves Carneiro Jr. (2006) *AJAX made simple with DWR.* JavaWorld.com

The Book of Dojo
http://dojotoolkit.org/book/dojo-book-1-0 [2008-04-14]

Introduction to DWR (Direct Web Remoting)
http://www.javapassion.com/ajax/DWR.pdf [2008]

Why choose Dojo?
http://ajaxian.com/by/topic/dojo/ [2007-07-09]

The Yahoo! User Interface Library
http://simon.incutio.com/slides/2006/xtech/yui-notes.html [2006-05-16]

Surveying open-source AJAX toolkits
http://www.infoworld.com/article/06/07/31/31FEajax_3.html [2007-07-31]

AJAX tutorial
http://www.w3schools.com/Ajax/Default.Asp

Direct Web Remoting
http://directwebremoting.org/

DOJO the javascript toolkit
http://dojotoolkit.org/

The Yahoo! User Interface Library (YUI)
http://developer.yahoo.com/yui/

JavaServer Pages Technology
http://java.sun.com/products/jsp/

JavaScript tutorial
http://www.w3schools.com/jS/

Enterprise JavaBeans Technology
http://java.sun.com/products/ejb/

JavaForum
http://javaforum.se

Appendix A – Definitions, Acronyms and Abbreviations

AJAX – Asynchronous JavaScript and XML
EJB – Enterprise Java Beans
CRUD – Create Read Update Delete
CSS – Custom Style Sheet
DOJO – JavaScript library
DWR – Direct Web Remoting
Errandhandler / Casebuild – the new application which is to be developed
ER – Entity Relation
HTML – HyperText Markup Language
IE – Internet Explorer
JS – JavaScript
JSP – Java Server Pages
RIA – Rich Internet Applications
SOAP – Simple Object Access Protocol
SQL – Structured Query Language
UI – User Interface
YUI – Yahoo User Interface
XML - eXtensible Markup Language

Appendix B – Requirement Specification

KRAVSPECIFIKATION 20071201

Casebuild

Sammanfattning

Underliggande arkitektur för Casebuild ska ändras, för att anpassas för asynkrona anrop och därmed för att öka användarvänligheten. Samtidigt är högsta prioritet att det ska vara lätt att underhålla och att all JavaScriptkod är snyggt uppdelad och lätt att förstå.

Tidslägen, operationer och dylikts ska kunna sparas utan att sidan laddas om.

Innehållsförteckning

1 VERSIONS HISTORIK

Revision	Date	Description
0.1	*2007-12-01*	*First issue. / Emanuella Wallin*

2 INTRODUKTION

2.1 SYFTE

Syftet med denna CR är att byta ut underliggande arkitektur för den del i WMS som kallas för Casebuild. Dels för att det idag krävs en full sidladdning för varje ändring som görs på sidan och dels för att undersöka huruvida ett javascript bibliotek kan användas i befintlig arkitektur.

3 PRELIMINÄR TIDSESTIMERING

320 h

4 KRAV

4.1 KRAV 1 – FILTRERING AV ÄRENDETYP EFTER GRUPP

4.1.1 KRAV 1 KOMMENTAR

En filtrering som filtrerar ärenden efter dess grupp ska finnas. Ett val där man kan se alla ärenden ska också finnas med, alltså en ”alla grupper” ska finnas med.

4.1.2 BEGRÄNSNINGAR

Ett ärende måste ingå i endast grupp.

4.2 KRAV 2 – SKAPA ETT NYTT ÄRENDE

4.2.1 KRAV 2 KOMMENTAR

Nya ärenden ska kunna skapas och sparas. Obligatoriska fält är:

- Ärendetypens namn
- Beskrivning
- Grupp

4.2.2 BEGRÄNSNINGAR

4.3 KRAV 3 – KONTROLLERA ÄRENDET MOT EN UTVALD GRUPP

4.3.1 KRAV 3 KOMMENTAR

En kontroll med ärendet mot en egen utvald grupp (ej grupp i samma betäckning som i krav beskrivna ovan). Den grupp som ärendet ska kontrolleras mot är de ärenden som har valts till höger i en av två listor.

4.3.2 BEGRÄNSNINGAR

4.4 KRAV 4 – KONTROLL AV ÄRENDET MED DATUMSTÄMPEL

4.4.1 KRAV 4 KOMMENTAR

För att ange en datumstämpel på valda ärenden i kontrollgruppen ska det finnas en radiobutton tillhörande till varje ärende i den högra listan.

4.4.2 BEGRÄNSNINGAR

4.5 KRAV 5 – KOMMENTAR FÖR VARJE ÄNDRING

4.5.1 KRAV 5 KOMMENTAR

En kommentar för varje ändring som görs på varje ärende ska göras. En dialog ruta ska visas där ändvändaren ska kunna skriva in en kommentar om ändringen som har gjorts på ärendet. Kommentaren, användarenamnet samt ett datum ska sparas till följd av detta för loggning i databasen. Bakomliggande information ska gråas ut och inte gå att editeras tills kommentaren är ifylld och sparad.

4.5.2 BEGRÄNSNINGAR

Det kommer endast att krävas en kommentar vid skapandet av ett nytt ärende enligt krav 4.2. En kommentar för varje ändring i relationer och operationer fliken ska göras.

4.6 KRAV 6 – LÄGGA TILL METODER PÅ ÄRENDE

4.6.1 KRAV 6 KOMMENTAR

Ska visas på en egen del som tillhör ”Tidslägen och Operationer”. Metoderna som tillhör ärendet ska visa i en lista i form av ett träd och kommer att vara statiskt.

4.6.2 BEGRÄNSNINGAR

4.7 KRAV 7– LÄGGA TILL AKTIVITET PÅ ÄRENDE

4.7.1 KRAV 7 KOMMENTAR

Under det träd som uppstår från krav 4.6 ska en lista med trädstruktur innehållande aktiviteter/tidslägen visas. Genom att trycka på en ikon till höger om rubriken ”Aktivitet (antal operationer)” ska en ny aktivitet skapas. Den nya aktiviteten ska placeras sist i listan

4.7.2 BEGRÄNSNINGAR

4.8 KRAV 8– SORTERING AV AKTIVITETER

4.8.1 KRAV 8 KOMMENTAR

Aktiviteter ska kunna sorteras med pilar och resultera i att ordningen ändras i databasen. Ordningen av aktiviteter bestämmer följden av tidslägena/aktiviteterna som ska utföras på ärendet. Varje ändring av ordningen, alltså varje klick på pilarna, ska resultera i att en kommentardialog visas och kräva en kommentar från användaren i enlighet med krav 4.5.

4.8.2 BEGRÄNSNINGAR

4.9 KRAV 9– LÄGGA TILL OPERATIONER PÅ AKTIVITET

4.9.1 KRAV 9 KOMMENTAR

Genom att trycka på en ikon till höger om aktivitet ska en ny operation skapas. Den nya operationen

4.9.2 BEGRÄNSNINGAR

4.10 KRAV 9– SORTERING AV OPERATIONER

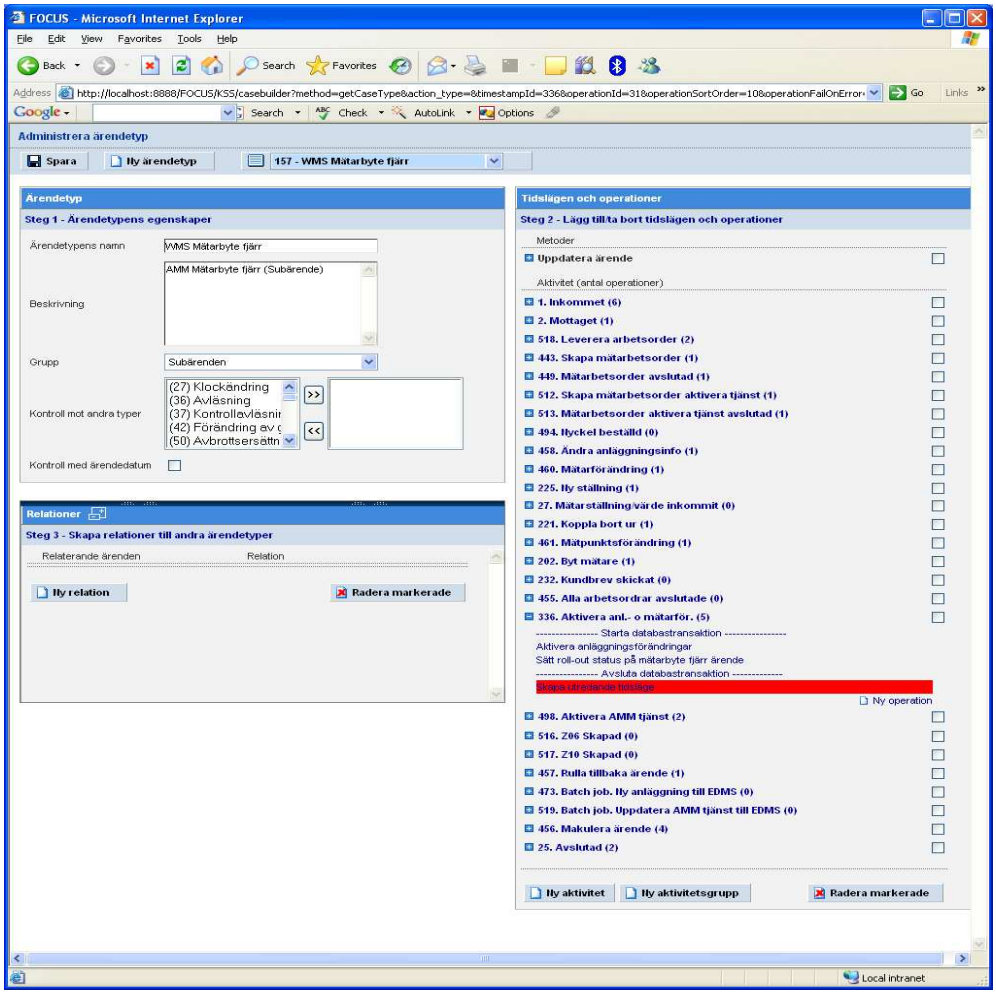
4.10.1 KRAV 9 KOMMENTAR

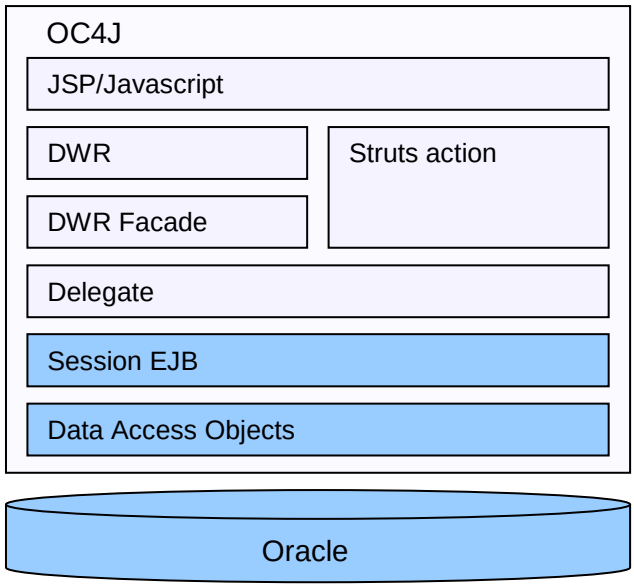
Operationer ska kunna sorteras med pilar och resultera i att ordningen ändras i databasen. Varje ändring av ordningen, alltså varje klick på pilarna, ska resultera i att en kommentardialog visas och kräva en kommentar från användaren i enlighet med krav 4.5.

4.10.2 BEGRÄNSNINGAR

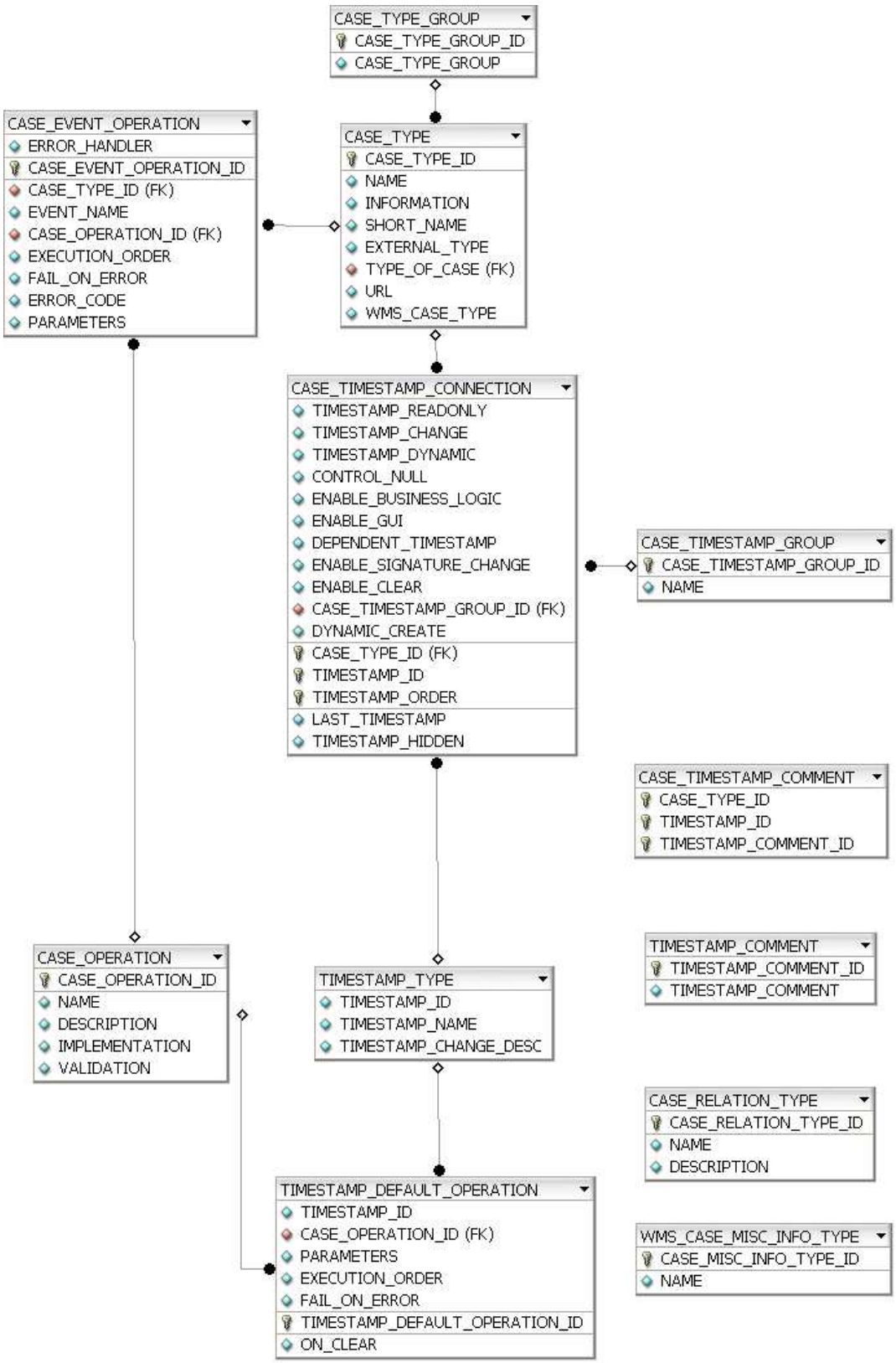
5 REFERENCES

Nedan visas ett exempel på en urvalsrapport, samt ett UM-underlag.





Appendix C - Database Diagram



Appendix D – Sourceforge.net – feature request

YUI Library

[1860983] [#2375428] Custom CSS for Dialog buttons

Submitted By:
Ella - emanuella

Date Submitted:
2007-12-30 16:41

Changed to Closed status by:
sdezzi

Closed as of:
2008-12-12 23:18

Last Updated By:
sdezzi - Comment added

Date Last Updated:
2008-12-12 23:18

Number of Comments:
2

Number of Attachments:
0

Category: (?)
Container - Dialog

Group: (?)
None

Assigned To: (?)
Satyen Desai

Priority: (?)
5

Status: (?)
Closed

Summary: (?)
[#2375428] Custom CSS for Dialog buttons

Private: (?)
No

It would be nice if it was possible to specify a CSS class for a button when creating a YAHOOO.widget.Dialog. The reason would be e.g. to provide different button icons (Cancel/Save) or to align them other than the default.

The CSS class could be specified in a new property, e.g. called "styleClass", as in this example:

```
YAHOO.example.container.myDialog =
new YAHOO.widget.Dialog("myDialog",
{ width:"300px",
buttons: [
{ text:"Cancel", handler:handleCancel,styleClass:"dialogCancel"},
{ text:"Save", handler:handleSave, isDefault:true,
styleClass:"dialogSave" }
]
});
```

I have already made local customisations to "configButtons: function (type, args, obj)" found in container.js to make this work:

```
for (i = 0; i < nButtons; i++) {
oButton = aButtons[i];

..

// Customisation begin
if (oButton.styleClass != null) {
oButtonEl.className += " " + oButton.styleClass;
}
// Customisation end

..

}
```

This is perhaps not the optimal way of solving it, but it works for me and

I think others could benefit from this feature as well.

Add a Comment:

Please log in!

Tracker items submitted anonymously should include a valid email address in the detailed description field. You will not receive notification of changes to Tracker items submitted anonymously.

DO NOT enter passwords or other confidential information!

SUBMIT

Followups:

Comments

Date: 2008-12-12 23:18
Sender: sdezzi
Hi Ella,

Thanks for the post. Based on feature prioritization, marking this one "wontfix". Container currently provides a way to walk through the buttons (through getButtons), allowing you to access the created buttons and add the classes if required.

Regards,
Satyen

Date: 2007-12-31 06:27
Sender: gpuckett
Logged In: YES
user_id=1793215
Originator: NO

Forwarding to component owner for consideration.

Attached Files:

Name	Description	Download
No Files Currently Attached		

Changes:

Field	Old Value	Date	By
close_date	-	2008-12-12 23:18	sdezzi

status_id	Open	2008-12-12 23:18	sdezzi
summary	Custom CSS for Dialog buttons	2008-11-21 08:13	gpuckett
assigned_to	nobody	2007-12-31 06:27	gpuckett