UBIWARE Deliverable D3.2:

Industrial Cases

April, 2010

Date	April 20, 2010
Document type	Report
Dissemination Level	UBIWARE project consortium
Contact Author	Vagan Terziyan
Co-Authors	Oleksiy Khriyenko, Sergiy Nikitin, Michal Nagy, Atte Pulkkis
Work component	WP7
Deliverable Code	D3.2
Deliverable Owner	IOG, JYU
Deliverable Status	Mandatory, Internal
Intellectual Property Rights	Unaffected

Table of Contents

UBIWARE Deliverable D3.2:
Workpackage WP7:

Introduction

The objective of this workpackage is to trial UBIWARE on real industrial cases. This has two major goals for such case studies. The first goal is to evaluate the scientific concepts behind UBIWARE and to find problems and issues in UBIWARE that would otherwise be overlooked. The second goal is to facilitate the further utilization of UBIWARE in the industry. Several specific cases, proposed by the industrial partners, are analyzed, designed and prototyped based on the UBIWARE platform. The reasons for prototyping are the same: to identify issues in UBIWARE that would get overlooked if the work was only theoretical and thus abstract, and to demonstrate the benefits of UBIWARE in a tangible way so to facilitate future industrial adoption.

In the 3rd project year we deliver three industrial cases, those of Fingrid, Inno-W and Metso Automation.

During the Year 3, with respect to all three cases the task is the following:

Task T3.1_w7: Developing a full prototype application: connecting to additional relevant resources and extending the interactions between them towards a sufficiently elaborated application.

UBIWARE Deliverable D3.2:
Workpackage WP7:

1 Fingrid case

1.1 Background

With respect to the UBIWARE approach and platform, Fingrid's main area of interest is in organizing smart data management related to the events/alarms which company gets from their control systems. Existing systems do not provide many possibilities for managing this data beyond storing it to a time-series log, and browsing it with some filtering possibilities. A wish is to that the data should get flexibly accessible, integrated with other related data, and possibilities should be provided for producing generalizing reports to the power system operation and asset management persons.

Fingrid has the following two databases that were so far the objects of interest in the UBIWARE's Fingrid case:

- **Event History database**: **Eventlog** (Oracle) in the office environment, to which data is automatically replicated from SCADA's event history database. A record in this database contains such information as the time of the event, event class, access area, substation ID, device ID, the state of the object, and some other.
- **Elnet database** (Oracle) that stores information about all the equipment, including circuit-breakers, disconnectors, transformers, capacitors, and other. A record in this database contains such information as device group, device ID, ownership (Fingrid or external), and other.

The unique device ID present in both databases enables join queries.

1.2 User's Guide

1.2.1 Introduction

This industrial case was developed by Industrial Ontologies Group for Fingrid Oyj as a part of UBIWARE project (2007-2010). This document primarily describes the improvements developed within the third year of the project. It includes only minimal documentation related to work performed in the first and second year of the project. This document is available as part of the final report and also as a standalone document integrated into application help section.

1.2.2 Proposals

In the beginning of the development phase, Fingrid specified these proposals:

Proposal code	Description
P1	Operation counts and operation time of compressors in compressed-air plants (Paineilmalaitosten kompressorien toimintakerrat ja käyntiaika)
P2	Operation time of compensation equipments (capacitors, reactors) and transformers (Kompensointilaitteiden ja muuntajien käyntiaika)
P3	Operation counts of circuit-breakers and disconnectors owned by Fingrid after the last maintenance (Katkaisijoiden ja erottimien toimintakerrat viime huollon jälkeen)
P4	Adding of new filtering conditions for equipment alarms of the job responsibility areas (R1 alarms) (Työaluekohtaisten laitehälytysten suodatusehtojen lisäys)
P5	Protection alarms to the experts of protection by email (Suojaushälytykset suojausasiantuntijoille)
P6	Tripping alarms will be sent more often than once a day, for example once a hour (Laukaisutiedot tiheämmällä lähetysvälillä)
P7	Developing of the user interface for management of filtering conditions (Käyttöliittymän kehittäminen suodatusehtojen hallintaa varten)

Table 1.1 – Fingrid proposals

Proposals P1 and P2 were being solved together and the result is available through page Operation_time_counts.html. Proposal P3 is in the waiting state due to establishment of testing environment. Proposals P4-P7 were solved together and the result is available through the administrator's interface, which will be described later.

1.2.3 Delivery

The application consists of one folder which contains the following subfolders:

Folder	Description
DB	Part of generic Ubiware platform (SAPL script database)
doc	Part of generic Ubiware platform (SAPL user's guide)
FinGrid	Data related to Fingrid industrial case
jedit	Part of generic Ubiware platform (jEdit syntax highlighting files)
lib	Part of generic Ubiware platform (3rd party JAR libraries)
ubiware	Part of generic Ubiware platform (Ubiware platform Java files)

Table 1.2 – Structure of the main case folder

Among these folders, the most important one is FinGrid. This folder contains these important files and folders:

Name	Folder	File	Description
admin	x		(NEW) Contains administrator interface related files (for more information go to section 1.2.8)
responsibilities	x		(NEW) Contains semantically annotated responsibilities (one per file)
wrapper	x		Contains configuration information related to an application that allows Fingrid case to be run as a service (wraps the whole case into one service)
lastCheck.dat		x	(Deprecated) This file used to contain date of the last responsibility check. Since this version the date of the last check is stored for each responsibility separately.
lastEvent.dat		x	(Deprecated) This file used to contain date of the last event that was sent. Since this version the date of the last event is stored for each responsibility separately.
Equipment_alarms.html		x	Shows general equipment alarms based on month, year, type, area, etc.
Event_groups.html		x	Shows events related to certain device types based on month and year.
index.html		x	Main page of the user interface
Operation_counts.html		x	Operation counts of certain equipment types based on month and year.
Operation_time_counts.html		x	(NEW) Solves proposals P1 and P2. Allows the user to see operation counts and total operation time of compressors, compensation equipment and transformers

Table 1.3 – Description of important files and folders within FinGrid folder

1.2.4 Hardware requirements

The server side containing agents needs a computer with the following minimal configuration:

Processor	AMD or Intel, single core, 1.6 GHz
Main memory (RAM)	1024 MB
Secondary memory space (HDD)	30 MB
Other requirements	Network connection

The client side consists of web interface (basic functionality and administrative interface). The basic functionality consists of very simple web pages with a few simple Javascript scripts. The minimal configuration for basic functionality only is the following:

Processor	Intel Pentium 2, 300 MHz
Main memory (RAM)	128 MB
Secondary memory space (HDD)	None
Other requirements	Network connection

The administrative interface contains more demanding Javascript functionality. Therefore more powerful computer is needed. In that case the minimal configuration is the following:

Processor	AMD or Intel, single core, 1.2 GHz
Main memory (RAM)	512 MB
Secondary memory space (HDD)	None
Other requirements	Network connection

1.2.5 Software requirements

The core of the application is written in Java programming language. There the server side needs to have installed Java runtime environment version 1.5 or higher. The client side needs to have a web browser with enabled Javascript support. The client side web application was developed for Internet Explorer 7. The recommended resolution is 1280x1024. In case of basic functionality only also 1024x768 is possible. However in case the user wants to use the administrative interface, he or she needs to have resolution at least 1280x1024, otherwise he or she will not be able to see the whole application pane.

1.2.6 Installation and execution

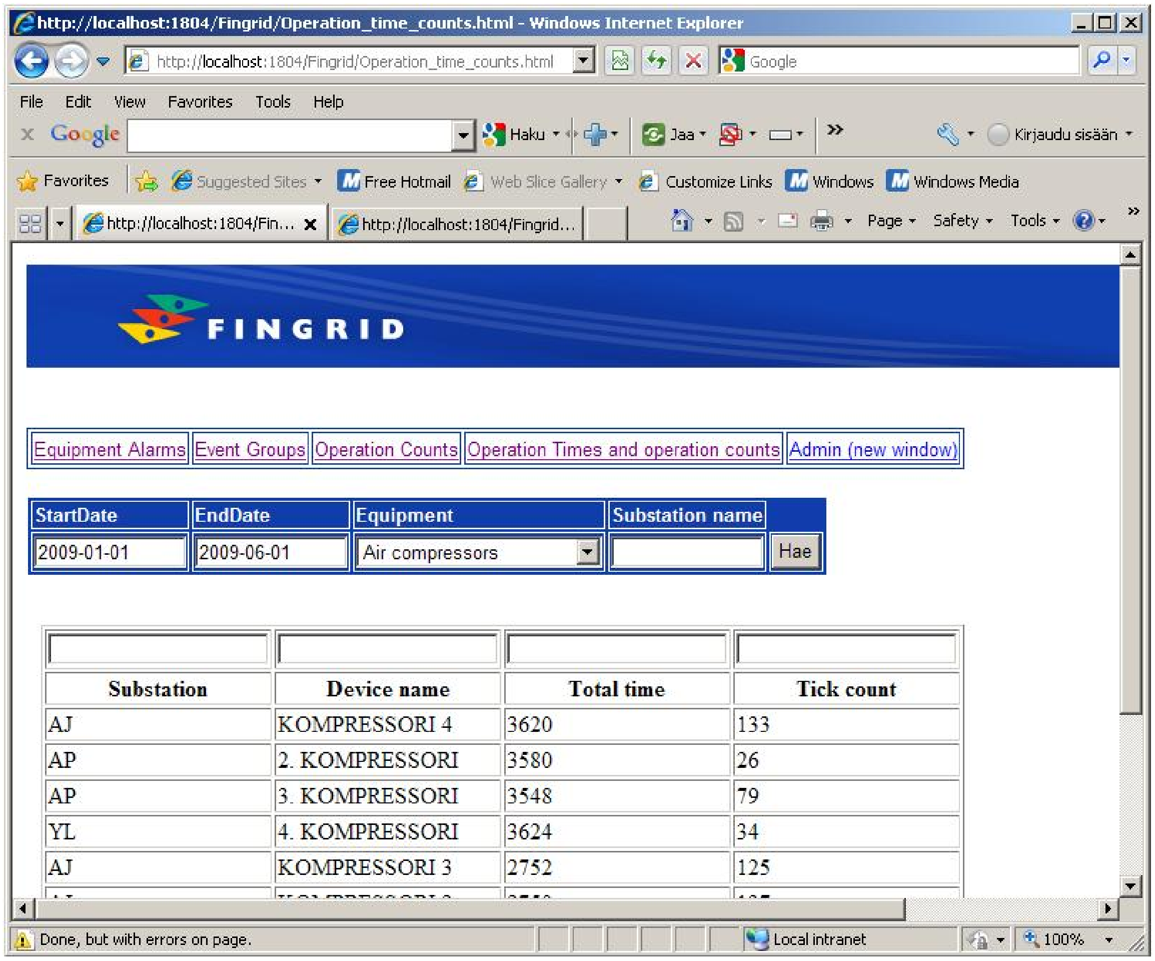
The server side installation consists of copying the application folder to any folder with read/write rights and starting three agents. One way to start the agents is to start the following scripts in this order:

- _run_FinGrid1.bat
- _run_FinGrid2.bat
- _run_FinGrid3.bat

Another way is to use so-called wrapper available from FinGrid folder. The wrapper will allow the user to install the application as a Windows service. In order to install the application, run script _InstallFingridWinService.bat from FinGrid folder. In order to start the service, run script _StartFingridWinService.bat.

1.2.7 Operation time and operation counts window

This feature was implemented to fulfill the proposals P1 and P2. It is implemented in file Operation_time_counts.html. The window with descriptions can be seen in Figure 1.1.


http://localhost:1804/Fingrid/Operation_time_counts.html - Windows Internet Explorer
File Edit View Favorites Tools Help
FINGRID
Equipment Alarms
Event Groups
Operation Counts
Operation Times and operation counts
Admin (new window)
Hae
Done, but with errors on page.
Local intranet
100%

StartDate	EndDate	Equipment	Substation name
2009-01-01	2009-06-01	Air compressors	

Substation	Device name	Total time	Tick count
AJ	KOMPRESSORI 4	3620	133
AP	2. KOMPRESSORI	3580	26
AP	3. KOMPRESSORI	3548	79
YL	4. KOMPRESSORI	3624	34
AJ	KOMPRESSORI 3	2752	125

Figure 1.1 – Operation times and counts screenshot

1.2.7.1 Filtering options

There are four filtering options. Three of them are mandatory and one is optional (Table 1.4).

Parameter name	Type	Description
StartDate	Mandatory	Start date of the time period being searched.
EndDate	Mandatory	End date of the time period being searched.
Equipment	Mandatory	You can choose which equipment time and counts will be displayed.
Substation name	Optional	Using a wildcard you can choose the substation(s) you want to display.

Table 1.4 – Filter description of operation times and counts

1.2.7.2 Result table

The result table contains four columns. The application allows the user to filter each column by specifying the searched string and pressing enter. The search is being performed by exactly matching the searched string. This means that if the user types AP above the substation header, only the rows with substation AP will be shown. Multiple search criteria are allowed. The meaning of the columns is as following:

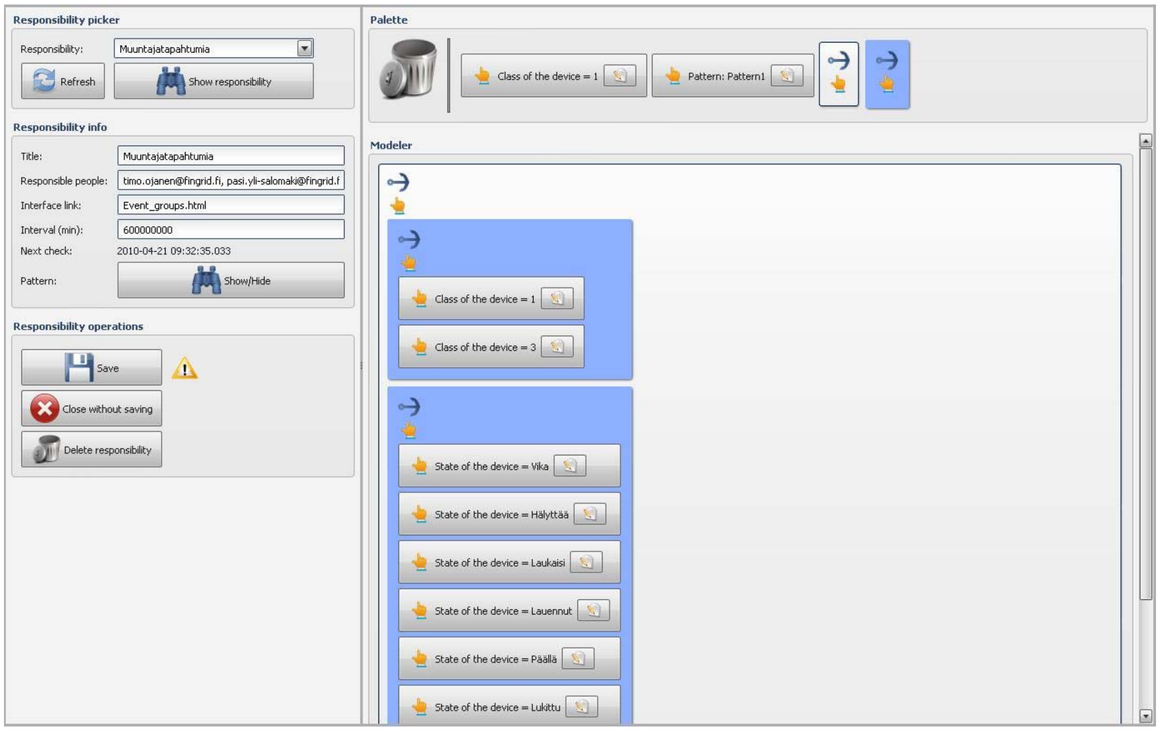
Column name	Description
Substation	Substation where the device is located
Device name	Name of the devices being examined
Total time	Total time of operation in hours
Tick count	Number of changes from OFF mode to ON mode

Table 1.5 – Column description of operation times and counts

1.2.8 Administrative interface

1.2.8.1 General information

The administrative interface was developed using a Javascript library/framework Qooxdoo 1.0.1. It is available through a web browser by clicking on link “Admin (new window)” from the main case page or from the top menu. This will open a new window containing the graphical user interface for responsibility editing. The interface can be seen in Figure 1.2.



Responsibility picker

Responsibility: Muuntajatapahtumia

Responsibility info

Title: Muuntajatapahtumia

Responsible people: timo.ojanen@fingrid.fi, pasi.yli-salomaki@fingrid.f

Interface link: Event_groups.html

Interval (min): 600000000

Next check: 2010-04-21 09:32:35.033

Pattern:

Responsibility operations

Palette

Class of the device = 1

Pattern: Pattern1

Modeler

Class of the device = 1

Class of the device = 3

State of the device = Vika

State of the device = Hälyttää

State of the device = Laukaisi

State of the device = Lauennut

State of the device = Päällä

State of the device = Lukittu

Figure 1.2 – User interface with modeler

1.2.8.2 Viewing and editing a responsibility

The administrative graphical interface consists of two main parts. On the left pane you can see controls and textual description of the responsibility. On the right pane there is so-called modeler with the palette.

The left pane consists of responsibility picker which allows the user to choose an existing responsibility or create a new one. When the application is started, the user can see only the responsibility picker area, which contains one combo box and three buttons (Figure 1.3). If the user wants to see an existing responsibility, first the user refreshes the view by clicking on “Refresh” button. This will contact the agent and ask him which responsibilities are available. After the agent’s answer has been received, the list of available responsibilities will be displayed in the combo box. The user then can choose a responsibility he or she wants to edit or delete. Upon clicking on the responsibility, two new areas (“Responsibility information” and “Responsibility operations”) will appear. The user can read and/or edit the responsibility information.

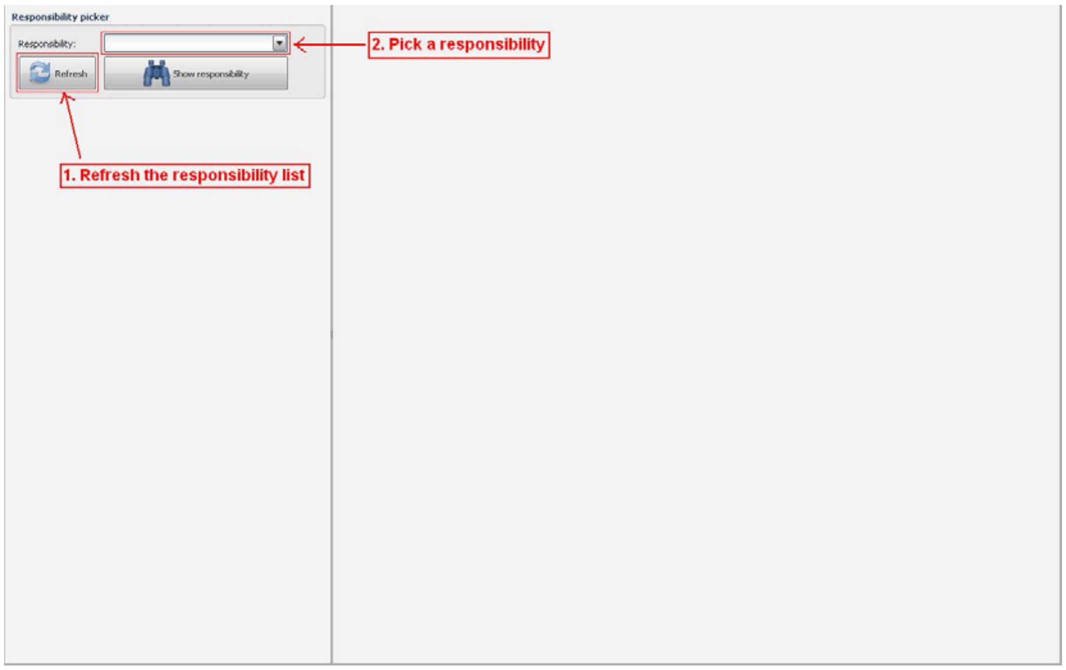


Figure 1.3 – Screenshot of the user interface upon first arrival

The Responsibility information area allows the user to edit any of the textual or numerical values related to the chosen responsibility. Every time a change is made, a small warning icon (yellow triangle) will appear next to Save button in Responsibility operations section. This means that some changes were made and the application will not allow the user to exit without explicitly confirming that the user doesn’t wish to save the changes.

The user can also click on Show/Hide button. This will open the right part of the user interface with the modeler. Modeler allows the user to see and possibly edit pattern by which the responsibility is defined. Modeler will be described later.

If the user is done with editing, he or she can press Save button to save the work. After the work has been saved a message dialog window confirming the save will appear. After that the user can close the responsibility and continue as if he or she started from the beginning. If the user didn't save the changes and he or she presses Close button, the application will ask if he or she really wants to close the responsibility without saving.

1.2.8.3 Modeling a responsibility pattern

In the right side of the interface you can find the modeler. The model consists of two parts. In the upper part there is a palette with available modeling elements. In the lower part there is the main modeler pane which is used to graphically display the logical expression being edited. The responsibility pattern is being modeled by dragging and dropping so-called logical elements into each other.

In general the drag and drop functionality work as following: Each element that is draggable has a handle and each element that is droppable has an anchor (Table 1.6). The element can be moved by pressing on the handle, moving while holding the mouse button and releasing the mouse button while being above some anchor. This will move the source element to the element which is the parent of the handle.

The palette contains a trash bin and four logical elements. Anything being dropped on the trash bin will be deleted. The four logical elements are as following:

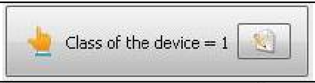
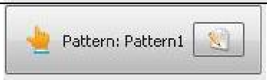
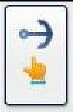
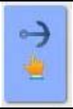
Name	Draggable	Droppable	Picture
Tripple statement	YES	NO	
Pattern statement	YES	NO	
AND container	YES	YES	
OR container	YES	YES	

Table 1.6 – Four modeler element types

Triple statement represents a simple statement in a form of a triple variable-sign-value. Variable is any of available parameters such as substation name, device class, device group, device name, etc. Sing is any logical sign expressing equality or inequality. Value is any value (numerical or string) the user wishes to add. Pattern statement allows the user to choose from a set of predefined patterns. Whenever the user uses this element, it means that the pattern chosen has to be fulfilled in order to consider this element to be of Boolean value true. Patterns are used to simplify the code and graphical representation by substituting a set of logical expressions just by one pattern statement. For example the user doesn't have to write a complicated rule for a capacitor in every rule deals with capacitors. Instead of that he or she can use this pattern. There are four patterns available:

Pattern name	Pattern meaning
Pattern_Capacitor	Any capacitor
Pattern_CircuitBreaker	Any circuit breaker
Pattern_Transformer	Any transformer
Pattern_Devices	Any device (capacitor or circuit breaker or transformer)

Table 1.7 – Predefined patterns available in the modeler

There are two elements which are containers. The function of AND container is to hold statements that have logical and connection between each other. In Figure 1.4 you can see an example of such statements. The expression can be understood as "Any device that belongs to substation AB and is of class 2 and is in state Vika". There is a rule that an AND container cannot contain another AND container. The application will not allow it, because every expression with an AND container in another AND container can be rewritten to a form with only one AND container.

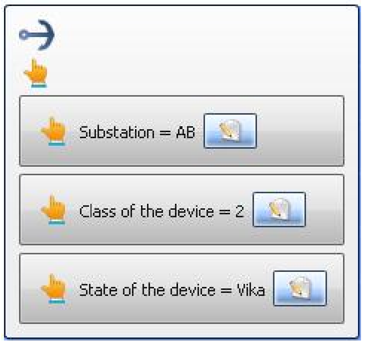


Figure 1.4 – An example of AND container

OR containers are very similar to AND containers, however they hold statements that have logical or connection between each other. The example from Figure 1.5 means "Any device that is of class 1 or 3". Same as in case of AND container there is a rule that an OR container cannot directly contain any other OR container, because it always can be rewritten in a form of just one OR container.

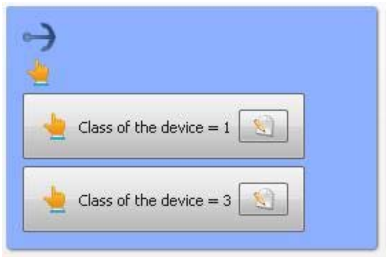


Figure 1.5 – An example of OR container

1.3 Developer's Guide

1.3.1 Introduction

This document is written for those who wish to understand this industrial case. Its main audience are programmers, there it will be more technical and written in a slightly less formal way. It is also a sort of a diary that can be used in the future to extend the industrial case with additional functionality. This documentation is not comprehensive, because the source code itself contains comments. This document is supposed to give the programmer an overall view of the problem.

It is recommended to read the user's guide first (at least till installation), because it contains information about proposals and other issues related to the case.

1.3.2 Technologies used

For the server side Java SE 1.5 was used. All three agents are Ubiware agents. The user interface uses HTML with Javascript in case of basic functionality and Qooxdoo platform 1.0.1 in case of administrative interface.

Qooxdoo is a Javascript framework that enables the user to write rich client applications. Some of the features are AJAX, cross-browser compatibility, compilation for optimization, etc. More information about qooxdoo can be found at http://qooxdoo.org/.

1.3.3 Operation counts and times

1.3.3.1 General information

This functionality fulfils the proposal P1 and P2. The idea is that there are certain types of devices (compressors, compensation equipment and transformers) and these devices can from time to time stop working. If this happens, the event is recorded (both stop and start event) and stored to Eventlog. The goal here was to create a tool that can will take as input certain time period and device type and as the result it will show for every device of that type how many times it went off (tick count) and what was the whole operational time (total time). This can be displayed as a table with table together with information from which substation the device comes from. As an additional feature the user should be able to search for devices only from certain substations. This is done by incorporating wildcard star (*), which means 0-n characters. Basically the searched string is taken and characters * are converted to % and then used in an SQL statements (substation like "ABC%"). If no text is put into substation field, then * is used, which means that all substations will be taken into consideration.

Short enumeration of inputs:

- Start date
- End date
- Device type (compressor, compensation equipment or transformer)
- Substation name (wildcards can be used)

Short enumeration of outputs (table columns):

- Substation
- Device name
- Total time (in hours)
- Tick count

Files involved:

- FinGrid/Operation_time_counts.html
- ubiware/fingrid/EvaluatorCompensationEq.java
- ubiware/fingrid/EvaluatorCompressor.java
- ubiware/fingrid/EvaluatorTransformer.java
- ubiware/fingrid/TimeCounterBehavior.java
- ubiware/fingrid/GenericEventSender.java
- ubiware/fingrid/IEvaluator.java
- FinGrid/FG_UI_OpTimeCount.sapl

Another requirement was to have a filter on top of the table. For this reason a free Javascript script from http://tablefilter.free.fr/ was used. It has MIT license, which goes well with the industrial case. This script allows you to enhance any HTML table with filtering functionality. It has a set of options that allow you to customize the appearance of the filter above the columns. Basically you just mark the desired table with an ID and then refer to the table using this ID when calling this table Javascript. An example can be seen here:

```
<script language='javascript' type='text/javascript' src='tablefilter.js'>
</script>
<table id="resultTable" >
 … bla bla bla …
</table>
<script language='javascript' type='text/javascript'>
      setFilterGrid("resultTable");
</script>
```

There is also an option to call setFilterGrid with two arguments, the second being an array of parameter-value pairs. This way you can customize the filter. Of course more information including nice examples can be found on the web page of the script.

1.3.3.2 Deeper description

The whole flow consists of these basic steps:
User chooses the search criteria and presses Hae
Data is parsed by Javascript on the page and it is transformed into SAPL (for more information read later)
Data is received by the agent
The agent takes the data and uses it as parameters for TimeCounterBehavior
Within this behavior based on the device type an action is performed
The result (already in HTML) is saved back to the agent in a form (?x :hasResponse ?resp)
Agent sends the result back normally by HttpResponseSenderBehavior

An example of HTTP call to the agent from the web page:

```
http://localhost:1804/:r01 :action "timeCount" . :r01 :hasParameters
{ :actionType :hasValue " compensationEquipment " . :startDate :hasValue
"2009-01-01" . :endDate :hasValue "2009-06-01" . :substation :hasValue
"AB*" }';
```

Out of these steps the most interesting one is probably the use of TimeCounterBehavior. As you may see from the code, GenericEventSender is acting as some kind of wrapper that wraps one of the evaluators. Evaluators are classes that implement IEvaluator interface. GenericEventSender connects to the database, performs the query that the evaluator gave him. Before the query is executed, it calls evaluator.start(). This will allow the evaluator to prepare data before the query has been executed. Then after the query has been executed, it does the following: For every row retrieved it calls evaluator.sendEvent(rset) and at the end it calls once evaluator.end(). There are 3 evaluators – each for different device type. The evaluator is looking for rows that involve information about ON and OFF state for a particular device. It records these states for each device separately and at the end it provides the summary such as total operation time and tick count.

Normally the device history should contain a series of alternating events ON, OFF, ON, OFF, etc. However, sometimes there might a problem and the sequence has ON-ON-OFF or ON-OFF-OFF. In case of ON-ON-OFF, we take the first ON and in case of ON-OFF-OFF we take the first OFF.

Another complication is the case when the user enters for example period 5.2. – 10.2 and the first event for a device X in this period is OFF. This means that some time in the past it was turned on. For these cases we take into account as the beginning of the interval midnight 5.2. Similar thing happens if the last even is ON. In that case we take 10.2. 23:59:59 as the end of the interval.

1.3.4 Administrative interface

1.3.4.1 General information

The front-end of this interface was developed in Qooxdoo and the back-end is written as a combination of SAPL script (FG_UI_Admin.sapl) and RABs. For more information on qooxdoo refer to the documentation on http://qooxdoo.org/documentation.

List of files involved – server side:

- FinGrid/ FG_UI_Admin.sapl
- ubiware/fingrid/ACLResponseSenderResponsibilitiesBehavior.java
- ubiware/fingrid/HttpResponseSenderPatternBehavior.java
- ubiware/fingrid/HttpResponseSenderResponsibilitiesBehavior.java

List of files involved – client side:

- Content of the folder: FinGrid/admin/custom/

Since qooxdoo is a new framework in this case, it is reasonable to describe it. It is an object-oriented framework/language which consists of classes stored in FinGrid/admin/custom/source/class/. The best way to understand qooxdoo is to go the home page of the project and read some tutorial. It will take 15minutes and you would have enough information to compile a project or edit some source code. One thing that is important is that in case you want to run the source version of the project, you need to have qooxdoo SDK installed and the path in the project has to point to this SDK. If you use build version, then you don't need qooxdoo SDK, however it doesn't show you any debug output.

1.3.4.2 Deeper description of the client side

Application.js:

This is the main class of the application. In its main method the graphical components are created and laid out. The schema can be seen in Figure 1.6.

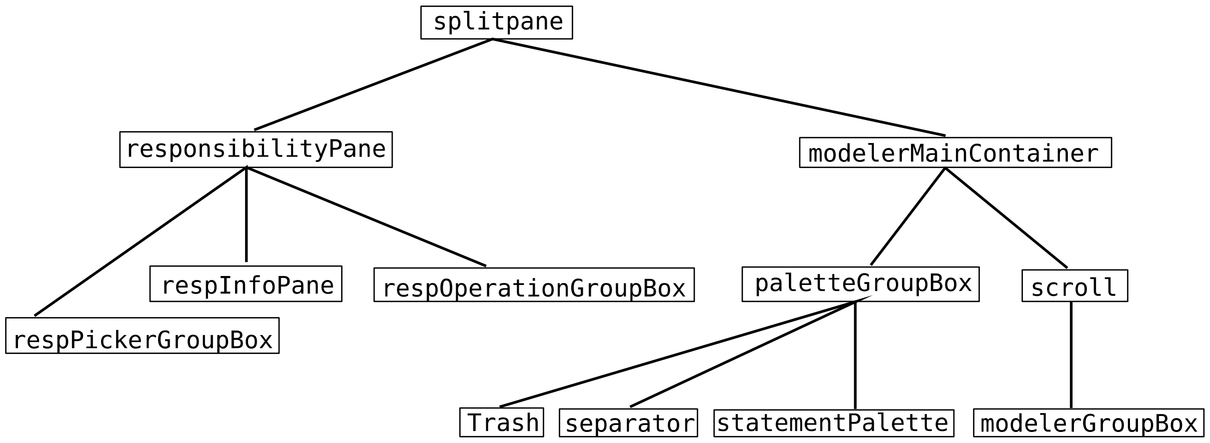


Figure 1.6 – Layout tree of graphical components within the administrative interface

The main method also contains event handlers and AJAX calls to the server side. The whole code within the main method is commented.

1.4 Future opportunities

One of the future possibilities is to develop a tool that would allow the user to create his/her own patterns in the administrative interface. Another interesting feature might be the ability of the agent to remember the history of all responsibilities. This would allow the user to return to any state from the past. It could work in a similar way as version control systems (CVS, subversion, etc.).

UBIWARE Deliverable D3.2:
Workpackage WP7:

2 Inno-W case

2.1 Background

The plan for the implementation of industrial cases for Inno-W Company was to elaborate adapter for a real RDF data storage of proposals to be browsed through 4I (FOR EYE) Browser in the context of close/similar resources. This case is a test bed for a research and development within the WP5.

2.2 Special requirements

Resource (proposals) descriptions from Inno-W's Databases should be adapted/transformed to the RDF format.

2.3 Inno-W industrial prototype

2.3.1 Adapter functionality and architecture

According to the general architecture of 4I (FOR EYE) Browser, adaptation of data sources is done via importing the source to the Browser and converting the data to the internal format. But it is not the only task for the adapter. In the same time with converting the original data format to internal one, adapter have to build all necessary visualization context descriptions and provide visualization module (MetaProvider) binding information.

All the adapters are developed as remote services. Input data for such services contains three parameters:

- ***"resultType"*** – type of the result (will be described later).There are two possible values: "*file_in_xml*" that corresponds to response in a form of XML String that contains the result content, and "*fileName_in_xml*" that requires response to be send as a set of file names in XML String;

- ***"servletURL"*** - URL of an adapter;
- ***"sourceFileURL"*** - URL of a source.

Current adapter works with both input formats: RDF with XML serialization and RDF in N-triple format. The data type of the source storage provided by Inno-W Company for the current industrial case is N-triple RDF.

An output depends on the Browser settings. As an output, Browser requires two options (as was mentioned before): all the result should be returned as a XML String, with the structure presented in the Figure 2.1; or XML String that contains file names (URLs) with correspondent content to avoid transferring of a huge amount of data (in this case adapter saves all the files on the own side)(see Figure 2.2). Further, this response from adapter will be parsed, converted resources storage will be saved locally and correspondent following files on the Browser side will be updated with a new data:

- *contexts.xml* – contains general descriptions of the contexts (IDs, names and set of resource classes they belongs to);
- *resClosenessContext.xml* – presents descriptions of resources closeness contexts (IDs, names, closeness calculation methods) including descriptions of contextual fields (field type, field property id and name, significance of the field and other field type dependent data);
- *metaprowiders.xml* – a set of available MetaProvider descriptions including (IDs, names, service addresses and requirements for input data with a respect to correspondent resource classes and visualization contexts);
- *defaultProfile.xml* – contains information about default visualization contexts for correspondent classes of resources and default MetaProviders for correspondent pairs of resources and visualization context.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<resources>
        … correspondent descriptions of the resources…
</resources>
<correspondent contexts>
        <closenessContexts>
                … correspondent closeness context  descriptions…
        </closenessContexts>
        <contexts>
                … correspondent general context  descriptions…
        </contexts>
        <mps>
                … correspondent descriptions of MetaProviders…
        </mps>
        <profile>
                … correspondent default profile data…
        </ profile>
</correspondent contexts>
```

Figure 2.1 – First type of response.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<fileNames>
        <fileName>
                … URL of the file that contains resources descriptions …
        </fileName>
        <fileName>
                … URL of the file that contains all other descriptions (Figure 2.1)…
        </fileName>
</fileNames>
```

Figure 2.2 – Second type of response.

Following xml structures shows us more detailed descriptions of all mentioned above parts of the response:

- **profile description**:

```
<profile>
  <contexts>
     <context forResource="resource class">context ID</context>
  </contexts>
  <metaProviders>
     <MetaProvider forResource="resource class" forContext="context ID">correspondent
           MetaProvider ID</metaProvider>
  </metaProviders>"
</profile>
```

- **metaprovider description;**

```
<mp>
    <mpId>correspondent MetaProvider ID </mpId>
    <name>MetaProvider name</name>
    <link>MetaProvider URL </link>
    <listener>field is not used in current implementation</listener>
    <rescont>
        <resClass>resource class</resClass>
        <cont>
            <contId>context ID</contId>
            <_in>resId</_in>
                    <_in>resLogo</_in>
                    <_in>resLogo_w</_in>
                    <_in>resLogo_h</_in>
                    <_ins resClass="resource class ">
                              <_in>resId</_in>
                              <_in>resLogo</_in>
                              <_in>resLogo_w</_in>
                              <_in>resLogo_h</_in>
                    </_ins>
                    <inResStorageXML>fileName</inResStorageXML>
                    <inClosenessContextXLM>fileName</inClosenessContextXLM>
                    <_out>resId</_out>
         </cont>
      </rescont>
  </mp>
```

- **general context description:**

```
<context>
    <contId>context ID</contId>
    <name>context name</name>
    <forClasses>
        <class>resource class</class>
    </forClasses>
</context>
```

- **closeness context description:**

```
<closenessContext>
    <closenessContext_id>context ID</closenessContext_id>
    <closenessContext_name>context name </closenessContext_name>
    <calculation_method>default method of general distance calculation
                                                            </calculation_method>
    <calculation_methods>
        <value>possible general distance calculation method</value>
        …
    </calculation_methods>
    <fieldContext>
          … field type dependent field context description (more details come later)…
    <fieldContext>
    …
</closenessContext>
```

- **resource description:**

```
<resource>
    <resId>resource ID</resId>
    <resClass>resource class</resClass>
    <name>resource name</name>
    <resTypeDes>
        <rtdItem>keyword for resource type description</rtdItem>
        …
    </resTypeDes>
    <resContDes>
        <rcdItem>keyword for resource content description</rcdItem>
        …
    </resContDes>
    <properties>
        <property>
            <prop_id>resLogo</prop_id>
            <prop_name>Logo</prop_name>
            <prop_value>URL of the resource logo (“?” – if none)</prop_value>
        </property>
        <property>
            <prop_id>resLogo_w</prop_id>
            <prop_name>Logo width</prop_name>
            <prop_value> width of the resource logo (“?” – if none)</prop_value>
        </property>
        <property>
            <prop_id>resLogo_h</prop_id>
            <prop_name>Logo height</prop_name>
            <prop_value> height of the resource logo (“?” – if none)</prop_value>
        </property>
        <property>
            … field type dependent resource property description …
        </property>
        …
    </properties>
</resource>
```

In the last two detailed descriptions (closeness context and resource descriptions) we have field type dependent context and property description. According to the distance measuring function (elaborated during the previous year as a part of visualization module that shows resource closeness), we consider 5 types of the properties/fields: *simple text field* (string), *keywords* (set of simple text fields), *complex text field* (finite set of text sub fields with the values from the defined set of them), *interval field* (with specified beginning and end of the interval) and field with *simple numerical* value.

There are following *fieldContext* and *property* descriptions for these 5 field types:

- **simple text field**:

```
<property>
    <prop_id>resource property ID</prop_id>
    <prop_name>resource property name</prop_name>
    <prop_type>textField</prop_type>
    <prop_value>value of the property</prop_value>
</property>

<fieldContext>
    <field_type>textField</field_type>
    <field_property_id>resource property ID</field_property_id>
    <field_property_name>resource property name</field_property_name>
    <field_significance>significance of the property</field_significance>
</fieldContext>
```

- **keywords field**:

```
<property>
    <prop_id>resource property ID</prop_id>
    <prop_name>resource property name</prop_name>
    <prop_type>keyWordsField</prop_type>
    <prop_values>
        <subValue>keyword or string</subValue>
            …
    </prop_values>
</property>

<fieldContext>
    <field_type>keyWordsField</field_type>
    <field_property_id>resource property ID</field_property_id>
    <field_property_name>resource property name</field_property_name>
    <field_significance>significance of the property</field_significance>
</fieldContext>
```

- **complex text field**:

```
<property>
    <prop_id>resource property ID</prop_id>
    <prop_name>resource property name</prop_name>
    <prop_type>complexTextField</prop_type>
    <subprop_names>
        <subprop_name>sub property name </subprop_name>
        …
    </subprop_names>
    <prop_values>
        <subValue>sub property value </subValue>
        …
    </prop_values>
</property>

<fieldContext>
    <field_type>complexTextField</field_type>
    <field_property_id>resource property ID</field_property_id>
    <field_property_name>resource property name</field_property_name>
    <field_significance>significance of the property</field_significance>
    <subprop_names>
        <subprop_name>sub property name </subprop_name>
        …
    </subprop_names>
    <corClasses>
        <corClass>
            <class_significance>significance of the sub property </class_significance>
            <value>sub property possible value</value>
            …
        </corClass>
        …
    </corClasses>
</fieldContext>
```

- **interval field**:

```
<property>
    <prop_id>resource property ID</prop_id>
    <prop_name>resource property name</prop_name>
    <prop_type>intervalField</prop_type>
    <subprop_names>
        <subprop_name>Interval beginning</subprop_name>
        <subprop_name>Interval end</subprop_name>
    </subprop_names>
    <prop_values>
        <subValue>value of the interval beginning</subValue>
        <subValue>value of the interval end</subValue>
    </prop_values>
</property>

<fieldContext>
    <field_type>intervalField</field_type>
    <field_property_id>resource property ID</field_property_id>
    <field_property_name>resource property name</field_property_name>
    <field_significance>significance of the property</field_significance>
```

```
        <subprop_names>
            <subprop_name>Distance between centers of the intervals</subprop_name>
            <subprop_name>Differences between lengths of the intervals</subprop_name>
        </subprop_names>
        <subField_significances>
            <value>significance of the distance between centers of intervals</value>
            <value>significance of the difference between length of intervals</value>
        </subField_significances>
        <field_calculation_method>default method of interval field distance calculation
                                                          </field_calculation_method>
        <field_calculation_methods>
            <value>possible interval field distance calculation method</value>
            …
        </field_calculation_methods>
    </fieldContext>
```

- **simple numerical field**:

```
    <property>
        <prop_id>resource property ID</prop_id>
        <prop_name>resource property name</prop_name>
        <prop_type>numberField</prop_type>
        <prop_value>value of the property</prop_value>
    </property>

    <fieldContext>
        <field_type>numberField</field_type>
        <field_property_id>resource property ID</field_property_id>
        <field_property_name>resource property name</field_property_name>
        <field_significance>significance of the property</field_significance>
    </fieldContext>
```

To simplify the process of *fieldContext* and *property* description creation we created a java class with the methods that add correspondent descriptions for those 5 types of the fields:

public String **addTextFieldProperty** (String *fieldID*, String *fieldName*, String *fieldValue*, String *field_significance*)

public String **addKeyWordsFieldProperty** (String *fieldID*, String *fieldName*, Vector<String> *fieldValues*, String *field_significance*)

public String **addComplexTextFieldProperty** (String *fieldID*, String *fieldName*, Vector<String> *subFieldNames*, Vector<String> *subFieldValues*, String *field_significance*, Vector<String> *subfield_significances*, Vector<Vector<String>> *subfield_possibleValues*)

public String **addIntervalFieldProperty** (String *fieldID*, String *fieldName*, String *intervalBeginning*, String *intervalEnd*, String *field_significance*, String *difCenter_significance*, String *difLength_significance*)

public String **addNumberFieldProperty** (String *fieldID*, String *fieldName*, String *fieldValue*, String *field_significance*)

2.3.2 Resource (Proposal) data structure

According to the provided by Inno-W ontology we selected a set of relevant properties that describe Proposal. This set includes 16 properties:

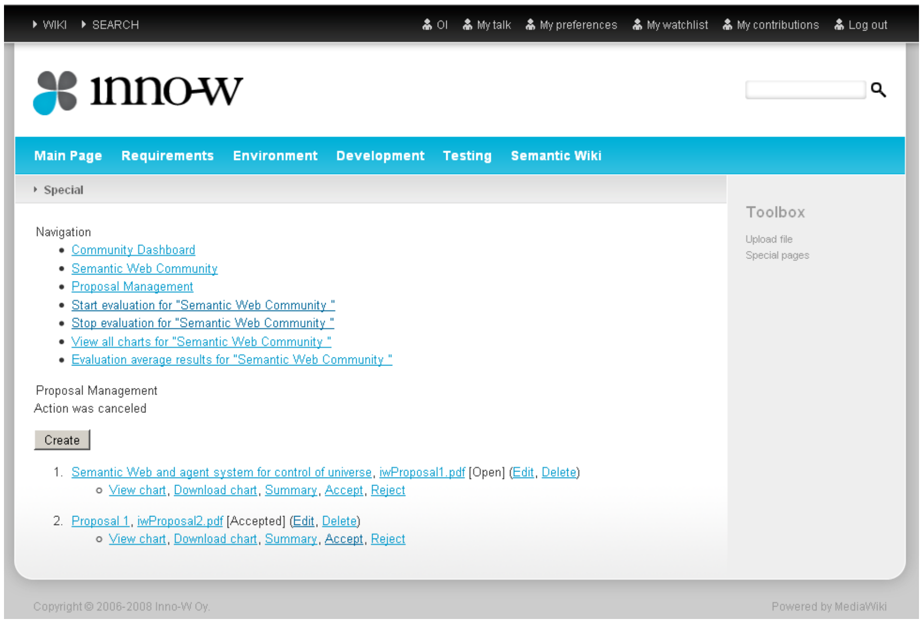
- **uri** – unique proposal textual identifier;
- **identity** – textual short name of the proposal;
- **name** – textual full name of the proposal;
- **status** – textual field that takes one of the predefined values from the following set {“open”, “submitted”, “undereval”, “evaluated”, “accepted” and “rejected”};
- **startYear** – field that contains 4 digits that present the start year of the project;
- **has_owner** – unique textual identifier of the proposal owner;
- **has_host_organization** – unique textual identifier of the host organization;
- **in_community** – unique textual identifier of the community proposal belongs to;
- **sizeMoney** – amount of money;
- **sizeMenYears** – amount of men years;
- **duration** – duration of the project in months;
- **numberOfPartners** – amount of partners in the project;
- **description** – textual field with a project description;
- **impact** – textual field with a project impact;
- **valueChain** – textual field with a set of related keywords;
- **networking** – textual field with a project networking description;

As we can see, there are mainly simple textual fields, a few numerical fields and only one *keywords field* – ***“valueChain”.*** To adapt and make the set of properties more useful/suitable and beneficial for distance measurement function, we modified and added some properties.

- **description** – *keywords field*, as a tokenized text from the value of former ***“description”*** property;
- **impact** – *keywords field*, as a tokenized text from the value of former ***“impact”*** property;
- **networking** – *keywords field*, as a tokenized text from the value of former ***“networking”*** property;
- **period** – *interval field*, with interval beginning (as a amount of months converted from the project start year) and interval end (as a sum of interval beginning and value from ***“duration”*** property);
- **Proposal_org_own_com** – *complex text field* with three sub fields (organization, owner and community). This combination could be useful if the values are correlated between each other. The possible values of the sub fields are collected from the available/used values of correspondent properties in the storage;
- **Proposal_org_own –** complex text field with two sub fields (organization and owner);
- **Proposal_org_com** – complex text field with two sub fields (organization and community);
- **Proposal_own_com –** complex text field with two sub fields (owner and community);

2.3.3 User's Guide

Inno-W Company provided us access to their portal where we can create sample set of proposals (see Figure 2.3). Following URL *"http://mvi.inno-w.com/triplify"* is used to retrieve the RDF source from the database.



Navigation

- Community Dashboard
- Semantic Web Community
- Proposal Management
- Start evaluation for "Semantic Web Community "
- Stop evaluation for "Semantic Web Community "
- View all charts for "Semantic Web Community "
- Evaluation average results for "Semantic Web Community "

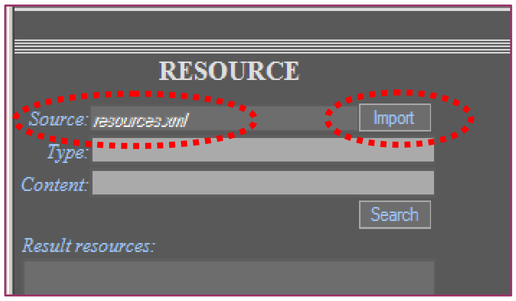
Proposal Management
Action was canceled

Create

1. Semantic Web and agent system for control of universe, iwProposal1.pdf [Open] (Edit, Delete)
   - View chart, Download chart, Summary, Accept, Reject
2. Proposal 1, iwProposal2.pdf [Accepted] (Edit, Delete)
   - View chart, Download chart, Summary, Accept, Reject

Copyright © 2006-2008 Inno-W Oy. Powered by MediaWiki

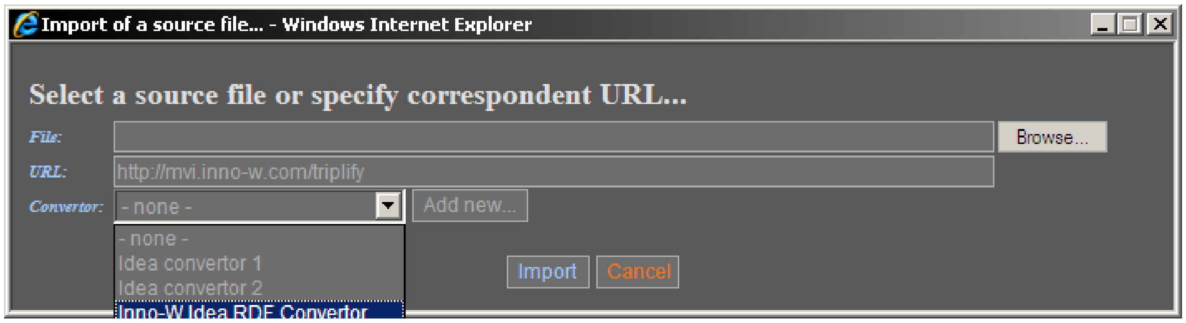
Figure 2.3 – Inno-W portal –editing of the proposals.



RESOURCE

Source: resources.xml

Button "Import" in the upper left part of 4I (FOR EYE) Browser is used to import certain source storage to the browser. In the popped up window (see Figure 2.4) user should specify link to the correspondent source. It can be URL or user can select the source from the local disc. At the same time user should select correspondent convertor from the list of available convertors supported by system.



Import of a source file... - Windows Internet Explorer

Select a source file or specify correspondent URL...

File:

URL: http://mvi.inno-w.com/triplify

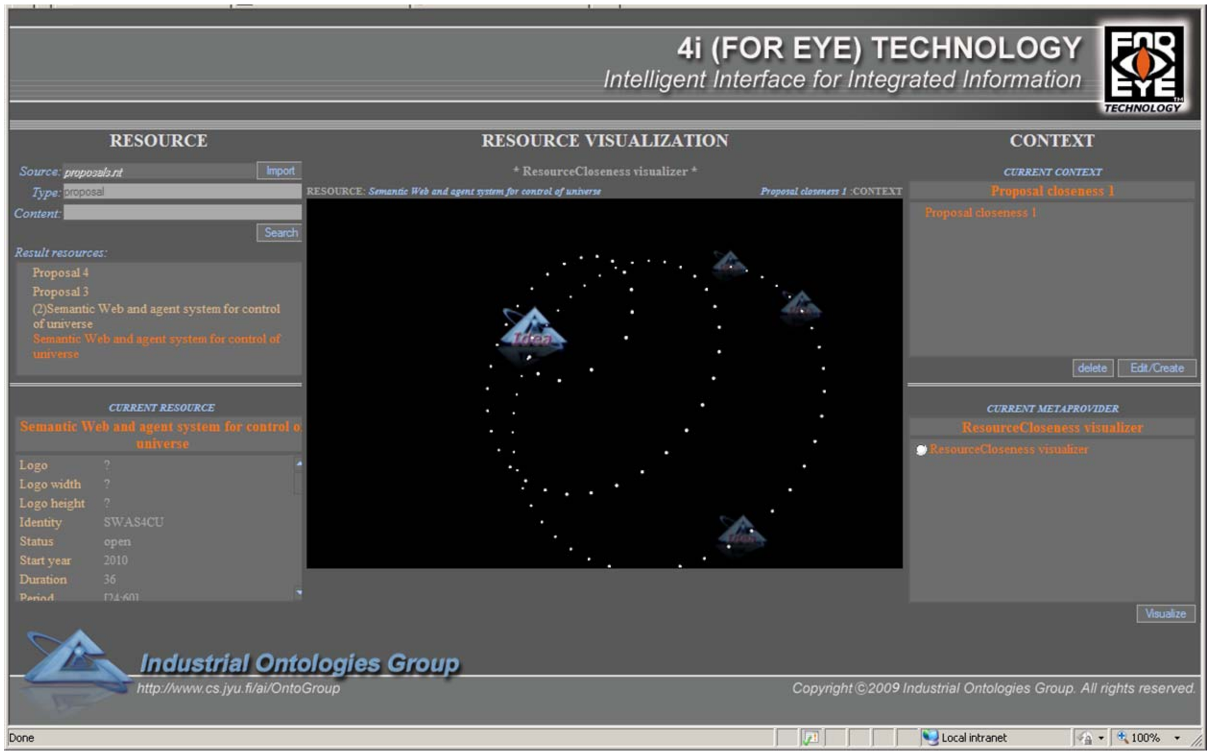
Convertor: - none -

- none -
Idea convertor 1
Idea convertor 2
Inno-W Idea RDF Convertor

Figure 2.4 – Import of the source storage to 4I (FOR EYE) Browser.

When user clicked the “Import” button on the popped up window, browser sends request to the selected convertor/adapter, gets response and update all the necessary files on its side. To indicate that new source has been imported, the value of the “Source” field in the upper left part of the browser is changed with a new source name.

Now (see Figure 2.5) user is ready to brows new data storage: search for the resources, visualize them in different contexts and create or edit those contexts. Each time when adapter is used, it creates one closeness visualization context for correspondent resource class(es) with some default values of the resource property significances to avoid the case when browser does not know any correspondent context for such resource class. Multiple import of the same source causes the addition the same default context (with different id). Further unnecessary/duplicated contexts can be deleted. Figure 2.6 shows us a window where we can edit or create new resource closeness visualization context for the proposals from the imported source.



Source: proposals.nt
Type: proposal
Content:

Result resources:
Proposal 4
Proposal 3
(2)Semantic Web and agent system for control of universe
Semantic Web and agent system for control of universe

CURRENT RESOURCE
Semantic Web and agent system for control of universe

Logo	?
Logo width	?
Logo height	?
Identity	SWAS4CU
Status	open
Start year	2010
Duration	36
Period	[24-60]

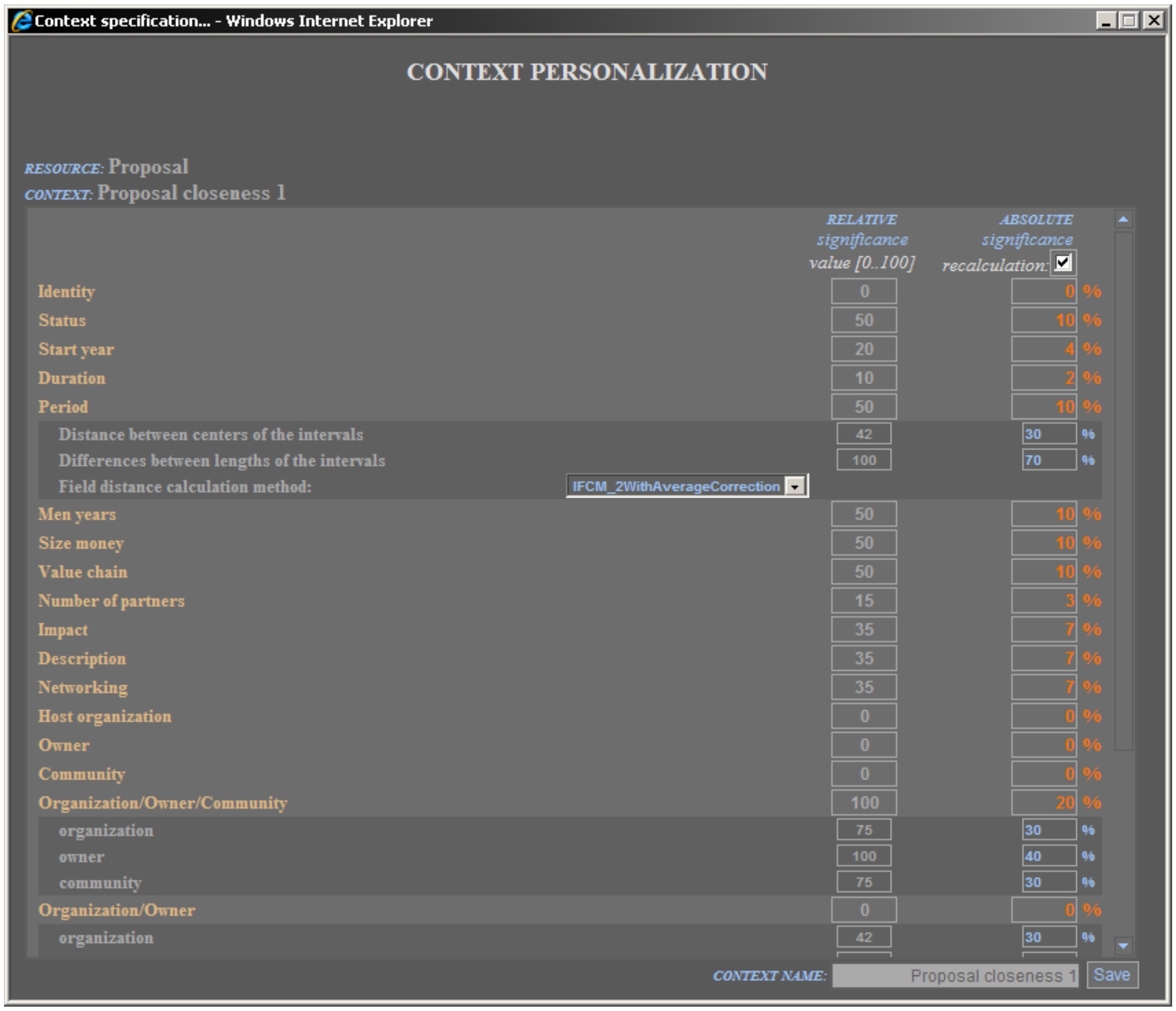
* ResourceCloseness visualizer *
RESOURCE: Semantic Web and agent system for control of universe
Proposal closeness 1 :CONTEXT

CURRENT CONTEXT
Proposal closeness 1
Proposal closeness 1

CURRENT METAPROVIDER
ResourceCloseness visualizer
ResourceCloseness visualizer

Copyright ©2009 Industrial Ontologies Group. All rights reserved.

Figure 2.5 – 4I (FOR EYE) Browser works with imported source.



Context specification... - Windows Internet Explorer

CONTEXT PERSONALIZATION

RESOURCE: Proposal
CONTEXT: Proposal closeness 1

	RELATIVE significance value [0..100]	ABSOLUTE significance recalculation: ✓
Identity	0	0 %
Status	50	10 %
Start year	20	4 %
Duration	10	2 %
Period	50	10 %
Distance between centers of the intervals	42	30 %
Differences between lengths of the intervals	100	70 %
Field distance calculation method: IFCM_2WithAverageCorrection		
Men years	50	10 %
Size money	50	10 %
Value chain	50	10 %
Number of partners	15	3 %
Impact	35	7 %
Description	35	7 %
Networking	35	7 %
Host organization	0	0 %
Owner	0	0 %
Community	0	0 %
Organization/Owner/Community	100	20 %
organization	75	30 %
owner	100	40 %
community	75	30 %
Organization/Owner	0	0 %
organization	42	30 %

CONTEXT NAME: Proposal closeness 1

Figure 2.6 – Resource closeness visualization context editing.

2.4 Future opportunities

Next step of the 4I (FOR EYE) Browser development will be concentrated on flexible possibility to add new adapters to the Browser and elaboration component based paradigm for further system development.

UBIWARE Deliverable D3.2:
Workpackage WP7:

3 Metso Automation case

3.1 Background

Metso industrial case was selected to be a test bed for a research and development within the WP2 (Distributed Querying and Integration) because of its “distributed nature” and big amounts of data in storages, that can not be collected in one place.

In the previous version of the industrial prototype we have integrated event flow data (events from monitoring and diagnostic systems) together with the structural and design data to provide a convenient assistant tool for an expert in diagnostics. In other words, ease the access to the relevant information needed for decision making.
We have integrated the most significant parts (derived from the use case) of the data samples provided. Those significant parts were semantically adapted. “Semantically adapted” means development of components, that represent the actual data sources as a virtual memory. I.e. the data from data sources was not fully transformed into S-APL, but it was annotated to answer the semantic queries instead. The description/annotation of the components refers to the domain ontology, which makes the specification explicit. At the same time, the descriptions themselves are not enough to run queries; they are interpreted by the Ontonuts engine and then used in planning and execution.

In the new version of the prototype we have made a manager for the components/Ontonuts developed. The manager provides functionality for editing and making test runs for the newly updated component descriptions.

3.2 Special requirements

The software requirements include a web browser and a Ubiware platform.

3.3 Metso Automation Prototype

The "Agent Component Manager" is developed as a web-based user interface that interacts with the UBIWARE platform. The interface is constructed with HTML and JavaScript technologies, using Qooxdoo[1] open-source JavaScript framework. The UI communicates with the UBIWARE platform via HTTP, in particular with the agent whose components are being managed.

3.3.1 User's Guide

The User Interface has three tabs (see Figure 3.1):

In the first tab, named "Browse/Edit", user is able to search and configure/update the existing components. At the moment the interface is tailored to certain type of components – Ontonuts. Or, being more precise, to the Donut type of the Ontonut, which is a semantic database connector. The user can edit the component in a nutshell, e.g. update the database query that is going to be executed, change the database URL, port number or username and password.

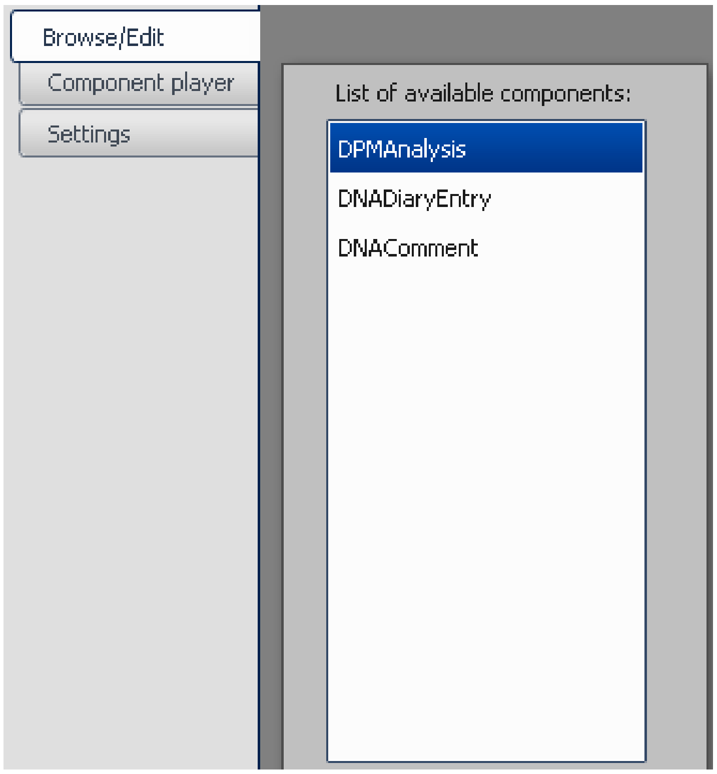


Figure 3.1 – The manager tabs.

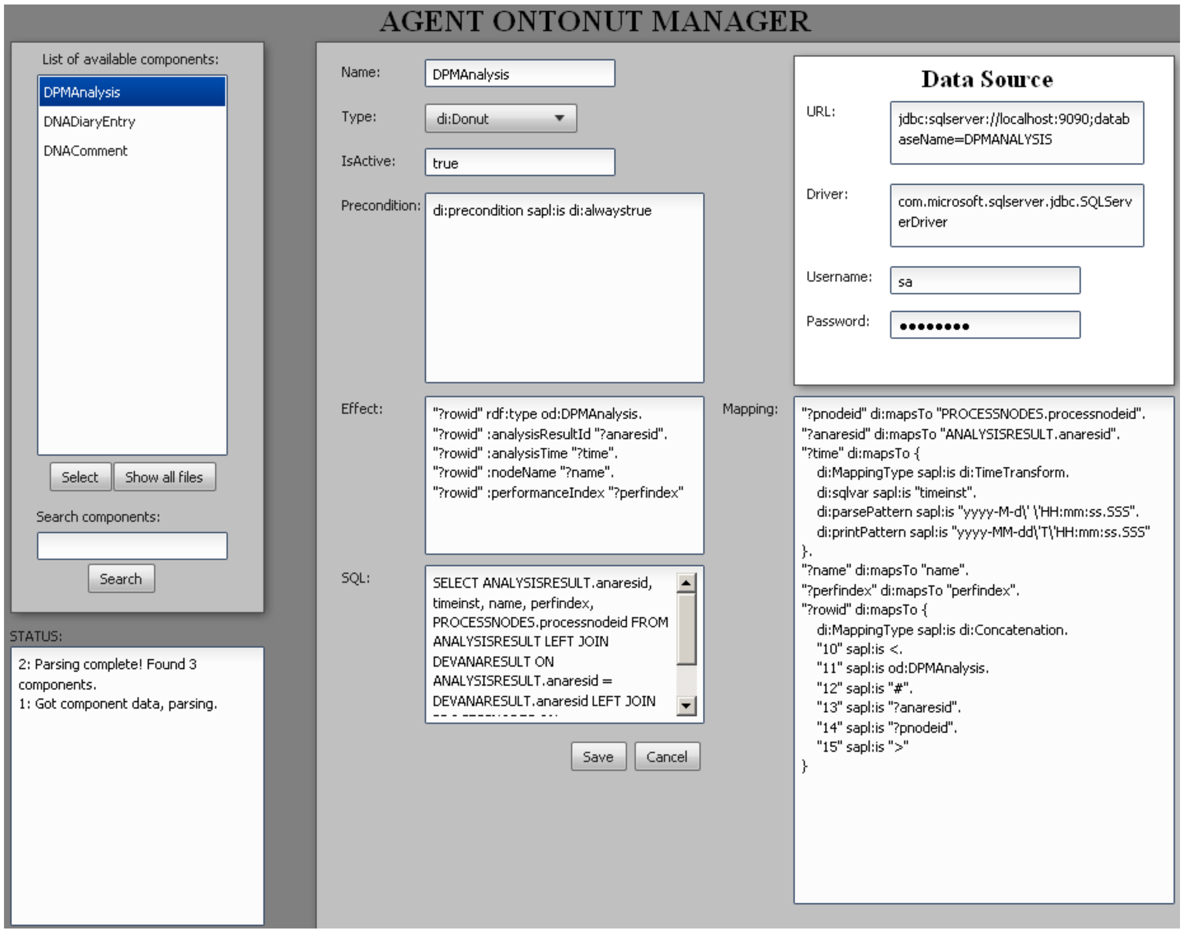
[1] www.qooxdoo.org

In second tab, named “Component Player”, the UI allows the user to specify a call to the component by adding and removing conditions/constraints. The user can execute the component call and see the results.

The third tab is reserved for the Settings of the editor itself and will be developed in the future versions of the manager.

- Browse/Edit -

In the“Browse/Edit” tab the user can choose the component from the list of available ones and modify component properties (see Figure 3.2).



List of available components:
DPMAnalysis
DNADiaryEntry
DNAComment

STATUS:
2: Parsing complete! Found 3 components.
1: Got component data, parsing.

Name: DPMAnalysis
Type: di:Donut
IsActive: true
Precondition: di:precondition sapl:is di:alwaystrue

Effect:
```
"?rowid" rdf:type od:DPMAnalysis.
"?rowid" :analysisResultId "?anaresid".
"?rowid" :analysisTime "?time".
"?rowid" :nodeName "?name".
"?rowid" :performanceIndex "?perfindex"
```

SQL:
```
SELECT ANALYSISRESULT.anaresid,
timeinst, name, perfindex,
PROCESSNODES.processnodeid FROM
ANALYSISRESULT LEFT JOIN
DEVANARESULT ON
ANALYSISRESULT.anaresid =
DEVANARESULT.anaresid LEFT JOIN
```

Data Source
URL: jdbc:sqlserver://localhost:9090;databaseName=DPMANALYSIS
Driver: com.microsoft.sqlserver.jdbc.SQLServerDriver
Username: sa
Password: ●●●●●●●●●

Mapping:
```
"?pnodeid" di:mapsTo "PROCESSNODES.processnodeid".
"?anaresid" di:mapsTo "ANALYSISRESULT.anaresid".
"?time" di:mapsTo {
   di:MappingType sapl:is di:TimeTransform.
   di:sqlvar sapl:is "timeinst".
   di:parsePattern sapl:is "yyyy-M-d\' \'HH:mm:ss.SSS".
   di:printPattern sapl:is "yyyy-MM-dd\'T\'HH:mm:ss.SSS"
}.
"?name" di:mapsTo "name".
"?perfindex" di:mapsTo "perfindex".
"?rowid" di:mapsTo {
   di:MappingType sapl:is di:Concatenation.
   "10" sapl:is <.
   "11" sapl:is od:DPMAnalysis.
   "12" sapl:is "#".
   "13" sapl:is "?anaresid".
   "14" sapl:is "?pnodeid".
   "15" sapl:is ">"
}
```

Figure 3.2 – The Browse/Edit tab.

To start editing, click the list of available components to select one, and press “Select”-button. Only one component can be viewed and/or edited at the same time.
The component editing view provides a set of text fields that can be modified by the user. The fields are not yet checked against mutual compatibility and therefore should be modified carefully. For example, “SQL”-field should be written in valid SQL. Failing to do so may result in inability to execute the database queries properly. Also, fields “Precondition”,

"Effect" and "Mapping" should be written in valid SAPL. For help with SAPL-related problems, consult the SAPL guide.

Saving changes happens by clicking "Save"-button. Settings will not be saved if the page is refreshed, user selects a new component or user clicks the "Cancel"-button.

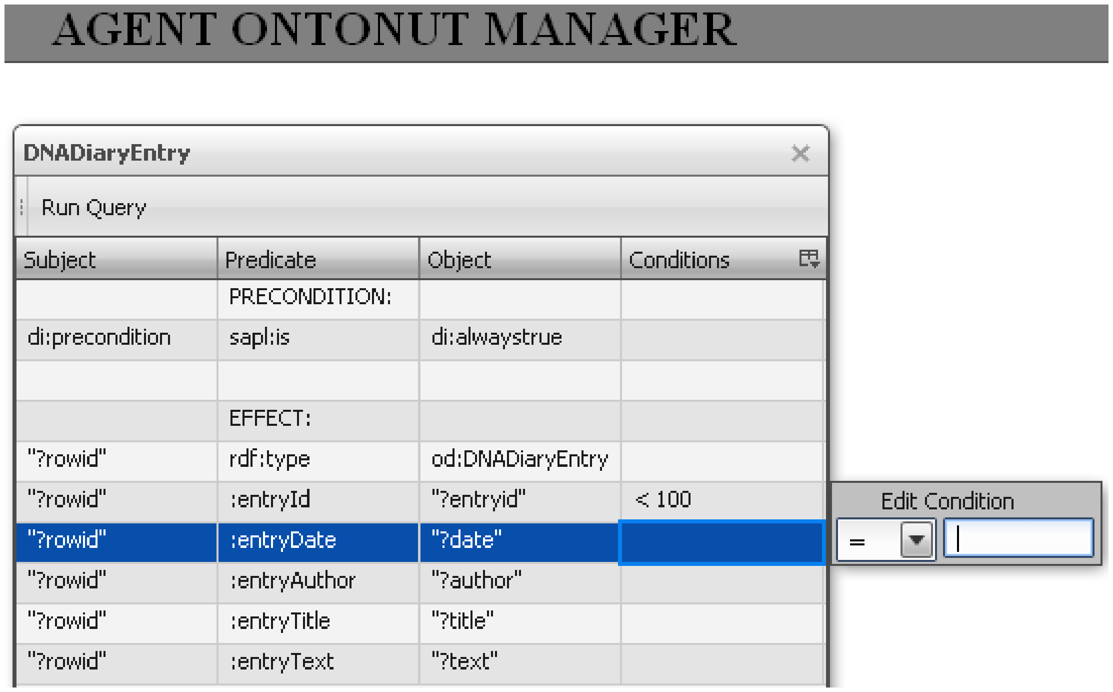
Cancelling changes can be done by clicking "Cancel"-button. Also, all unsaved settings will be discarded in event of a page refresh.

Searching through the list of available component can be done with "Search"-field located under the component list. Enter the name, or part of name, of component you want to find from the list, and click "Search" to initiate the search. Clicking the "Search"-button will filter out all items that do not match the search parameter.

Refreshing component list can be done in two ways. To un-filter the component list (filtering is done by "search"-function), click "Show all files" –button to show the un-filtered list of components. To force the interface to search for new components, refresh the UI page. Upon refresh, the UI will request new information from corresponding UBIWARE-agent.

- Component Player -

To start, you can either drag & drop or select & add a component from the list. Multiple components can be viewed at a time. Example component view is shown on Figure 3.3.



Subject	Predicate	Object	Conditions
	PRECONDITION:		
di:precondition	sapl:is	di:alwaystrue	
	EFFECT:		
"?rowid"	rdf:type	od:DNADiaryEntry	
"?rowid"	:entryId	"?entryid"	< 100
"?rowid"	:entryDate	"?date"	
"?rowid"	:entryAuthor	"?author"	
"?rowid"	:entryTitle	"?title"	
"?rowid"	:entryText	"?text"	

Figure 3.3 – The Component player tab: Call/Query specification.

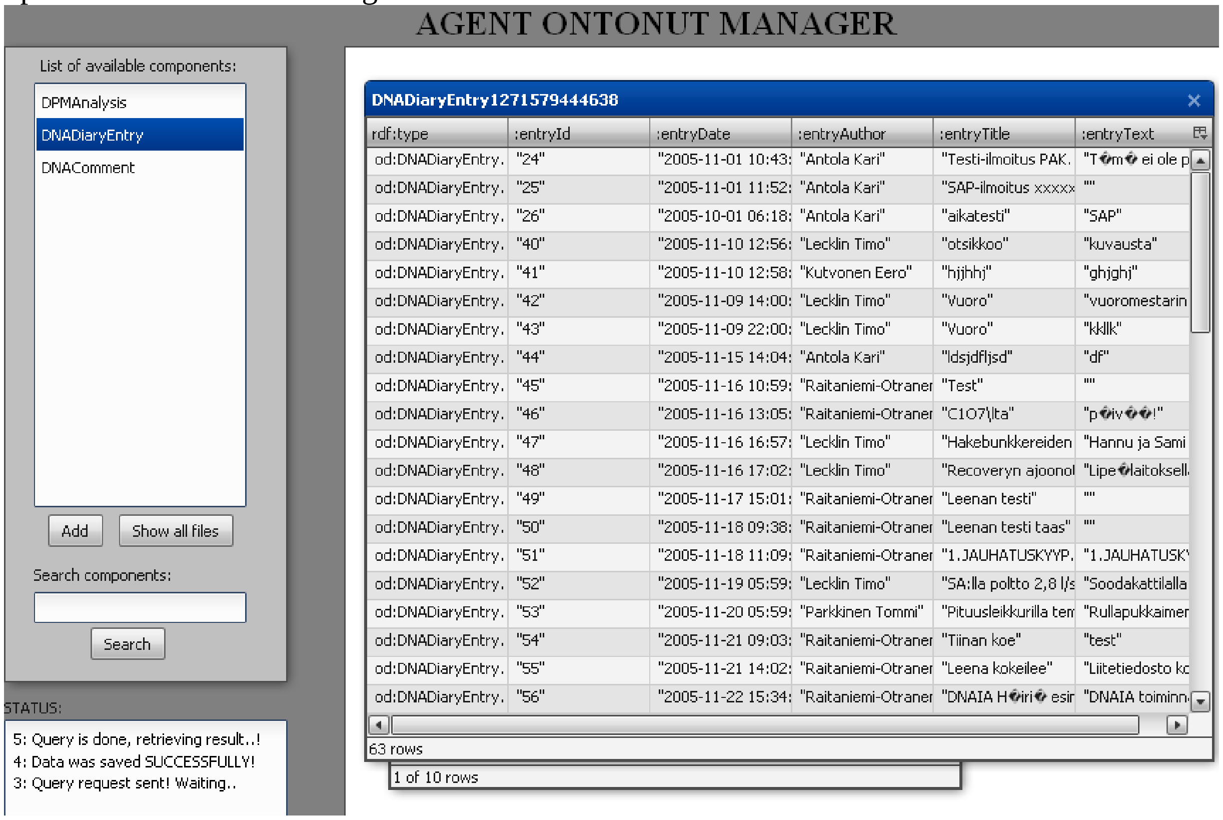
To “drag & drop”, push mouse button down on a component in the list, hold down the button while dragging mouse over the large white are, and release the button over the area. Doing this will create a window showing the component data in a window.
Second option is to click the list of available components and then push the “Add”-button. The components selected in the list will be added to the white field.

Searching through the list of available component can be done with “Search”-field located under the list. Enter the name, or part of name, of component you want to find from the list, and click “Search” to initiate the search. Clicking the “Search”-button will filter out all items that do not match the search parameter.

Editing can be done through the components window in the view.

Adding conditions is done by clicking any cell of the table in the window. A condition editing window pops up, and the condition type (> or < for example) and condition value (100, 200 for example) can be set. To finish editing the condition, click anywhere else on the table, and condition is saved.

Executing database query can be executed by clicking the “Run Query” –button, in the upper left corner of the component window. The query will be sent to agent in UBIWARE-platform, and it will run the query according to information given. After query is done and results processed, the result will be returned to UI and the “Component Player” –page will open a new window showing the results.



5: Query is done, retrieving result..!
4: Data was saved SUCCESSFULLY!
3: Query request sent! Waiting..

DNADiaryEntry1271579444638

rdf:type	:entryId	:entryDate	:entryAuthor	:entryTitle	:entryText
od:DNADiaryEntry.	"24"	"2005-11-01 10:43:	"Antola Kari"	"Testi-ilmoitus PAK.	"T�m� ei ole p
od:DNADiaryEntry.	"25"	"2005-11-01 11:52:	"Antola Kari"	"SAP-ilmoitus xxxxx	""
od:DNADiaryEntry.	"26"	"2005-10-01 06:18:	"Antola Kari"	"aikatesti"	"SAP"
od:DNADiaryEntry.	"40"	"2005-11-10 12:56:	"Lecklin Timo"	"otsikkoo"	"kuvausta"
od:DNADiaryEntry.	"41"	"2005-11-10 12:58:	"Kutvonen Eero"	"hjjhhj"	"ghjghj"
od:DNADiaryEntry.	"42"	"2005-11-09 14:00:	"Lecklin Timo"	"Vuoro"	"vuoromestarin
od:DNADiaryEntry.	"43"	"2005-11-09 22:00:	"Lecklin Timo"	"Vuoro"	"kkllk"
od:DNADiaryEntry.	"44"	"2005-11-15 14:04:	"Antola Kari"	"ldsjdfljsd"	"df"
od:DNADiaryEntry.	"45"	"2005-11-16 10:59:	"Raitaniemi-Otraner	"Test"	""
od:DNADiaryEntry.	"46"	"2005-11-16 13:05:	"Raitaniemi-Otraner	"C1O7|lta"	"p�iv��!"
od:DNADiaryEntry.	"47"	"2005-11-16 16:57:	"Lecklin Timo"	"Hakebunkkereiden	"Hannu ja Sami
od:DNADiaryEntry.	"48"	"2005-11-16 17:02:	"Lecklin Timo"	"Recoveryn ajoonol	"Lipe�laitoksell
od:DNADiaryEntry.	"49"	"2005-11-17 15:01:	"Raitaniemi-Otraner	"Leenan testi"	""
od:DNADiaryEntry.	"50"	"2005-11-18 09:38:	"Raitaniemi-Otraner	"Leenan testi taas"	""
od:DNADiaryEntry.	"51"	"2005-11-18 11:09:	"Raitaniemi-Otraner	"1.JAUHATUSKYYP.	"1.JAUHATUSK
od:DNADiaryEntry.	"52"	"2005-11-19 05:59:	"Lecklin Timo"	"SA:lla poltto 2,8 l/s	"Soodakattilalla
od:DNADiaryEntry.	"53"	"2005-11-20 05:59:	"Parkkinen Tommi"	"Pituusleikkurilla ter	"Rullapukkaimer
od:DNADiaryEntry.	"54"	"2005-11-21 09:03:	"Raitaniemi-Otraner	"Tiinan koe"	"test"
od:DNADiaryEntry.	"55"	"2005-11-21 14:02:	"Raitaniemi-Otraner	"Leena kokeilee"	"Liitetiedosto kc
od:DNADiaryEntry.	"56"	"2005-11-22 15:34:	"Raitaniemi-Otraner	"DNAIA H�iri� esir	"DNAIA toiminn

63 rows

1 of 10 rows

Figure 3.4 – The Component player tab: Call result.

3.3.2 Agent Component Manager API

This subsection contains information on the files, classes and methods used in ACM.

Application.js (acm.Application)
Extends: qx.application.Standalone

Description: acm.Application is the main class for creating and handling the User Interface for Agent Component Manager. This class acts as an interface between the Ubiware-platform and other User Interface functions.

Following sections will contain information on the specific methods of acm.Application.

main: function ()

Description: When .html file is called and the files loaded, this is the method that gets called first. The method enables debug consoles (if possible), creates UI elements and requests information from agent memory.
Parameters: ---
Return: ---

createUiObjects: function ()

Description: This method takes care of creating the UI elements. The method itself creates the tab view and its settings. Content for the pages is created in sub-methods *uiCreateBrowseEdit: function (targetTab)* and *uiCreateTables: function (targetTab).*
Parameters: ---
Return: ---

qooxdooHttpGet: function ()

Description: This is a method that queries agent beliefs. The method is called via event handlers.

searchComponents: function (target)

Description: Function *searchComponents* takes the list of component items and filters out all list items that do not match the description given.
Parameters:

• *target*	**String**	The parameter contains the substring given in the search-field

Return: ---

refreshItemList: function (pageNumber)

Description: Removes any filtering of the list items on the given page.

Parameters:

• *pageNumber*	**Int**	The array index where the list-to-be-refreshed is located

Return: ---

---

parseResponse: function ()

Description: Parses the data saved by *qooxdooHttpGet*. The parsing process is organized as follows:
1) Extract prefix information.
2) Make initial list of components.
3) Request the component descriptions from agent memory
4) Refresh item list
Parameters: ---
Return: ---

---

addStatusMessage: function (message)

Description: This method adds messages to the "status"-window on the current page.
Parameters:

• *message*	**String**	The message to be added.

Return: ---

---

selectComponent: function (selection)

Description: Extracts the component information from *acm.Component*-objects stored in the array, and puts it into corresponding UI elements.
Parameters:

• *selection*	**Int**	The index number of the component that was selected.

Return: ---

---

tempSave: function ()

Description: Updates changed component information to agent memory. The process is organized as follows (done for each property separately):
1) Check if the information has changed. If yes, continue.
2) Send request to remove old value(s) from agent memory.
3) Send request to save new value to agent memory.
Parameters: ---
Return: ---

---

qooxdooHttpSaveStatement: function (triple)

Description: Sends an HTTP request to save the SAPL-triple given to agent memory.
Parameters:

- *triple* **String** An SAPL-triple to be saved. Non-SAPL string will not be saved.

Return: ---

---

addTable: function (compIndex)

Description: This method is used to add tables (*acm.ComponentTable*) to table/query handling area. Table settings and event handlers are created here.
Parameters:

- *compIndex* **Int** The index of the component represented in the table.

Return: ---

---

saveQuery: function (sItem)

Description: Retrieves “effect” and “precondition” data from the component item given to build a component request. The request is then sent to the agent.
Parameters:

- *sItem* **acm.ComponentTable** The component item object.

Return: ---

---

convertSaplNamespace: function (text)

Description: This method converts all namespaces from the user-readable format into the Ubiware-processable format.
Parameters:

- *text* **String** The text to be converted.

Return:

- *String* The string that is returned is in Ubiware-processable format.

---

qooxdooHttpExecuteQuery: function (qId)

Description: Sends an HTTP request with the command to execute a query that already exists in the agent memory. By default, the method waits 200 seconds for a response with query results. The results received, are added into the window table/query management area.
Parameters:

- *qId* **Int** The ID number of the query to be executed.

Return: ---

---

qooxdooHttpRemoveStatement: function (statement)

Description: Sends an HTTP request with the command to remove beliefs according to the statement given.
Parameters:

- *statement* **String** The command according to which beliefs will be removed.

Return: ---

---

qooxdooHttpRequestStatement: function (compIndex, id, propertyTag, property)

Description: Sends an HTTP request to retrieve information from agent memory, based on parameters given. The method initiates other statement requests once the data source id has been determined.
Parameters:

- *compIndex* **Int** The array index of the component in question (for which information is being retrieved).
- *Id* **String** The id (=name) of the component. This is sent to agent as part of the request.
- *propertyTag* **String** An SAPL namespace-tag of the property we are looking for.
- *property* **String** The name of the property, the value of which we request.

Return: ---

Component.js (acm.Component)
Extends: qx.core.object

Description: Originally intended to serve as a parent class for different types of components. The class is not in use currently, since only one component type is available at the moment.

ComponentTable.js (acm.ComponentTable)
Extends: qx.ui.window.Window

Description: ComponentTable is a window-element containing a table and other functions for handling component information and component calls/queries.

Following sections contain the information on the specific methods of acm.ComponentTable

---

construct: function (label, effectPrecondData)

Description: The constructor for this class.
1) Sets default settings.
2) Sends the data to be parsed.
3) Creates the proper elements and event handlers.

Parameters:

- *label* **String** The label/caption that will be shown for this window.
- *effectPrecondData* **[String[], String[]]** effectPrecondData is a 2-dimensional array, where index [0] is an array with “effect” information and index [1] is an array with “precondition” information.

Return: ---

---

getPrecondData: function ()

Description: Extracts and returns precondition data from the table.
Parameters: ---
Return:

- *String* Returns a string containing the precondition data. The method may return empty string if no precondition data was found.

---

getEffectData: function ()

Description: Extracts and returns effect data from the table.
Parameters: ---
Return:

- *String* Returns a string containing the effect data. The method may return empty string if no effect data was found.

---

saveCondition: function (cond, x, y)

Description: Saves the given condition into the row and column specified.
Parameters:

- *cond* **acm.Condition** The condition element.
- *x* **Int** The row number of the cell, where the condition is saved to.
- *y* **Int** The column number of the cell, where the condition is saved to.

Return: ---

---

getCondition: function (x, y)

Description: The function tries to return the condition data for the cell specified. If none was found, return a new *acm.Condition* object.
Parameters:

- *x* **Int** The row number of the cell, the condition data is retrieved from.
- *y* **Int** The column number of the cell the condition data is retrieved from.

Return:

- *acm.Condition* If no condition for the given row and column was found, a new, empty, condition object window is returned. If any data was found, it is put into the *acm.Condition*-object and then returned.

---

getConditionData: function ()

Description: Extracts and returns the condition data stored in this object.
Parameters: ---

Return:

- *String* Returns a string containing the condition data. The method returns an empty string, if no condition data was found.

Condition.js (acm.Condition)
Extends: qx.ui.container.Composite

Description: acm.Condition is a condition editor window that is used to handle the transfer of condition data between user and UI.

Following sections will contain information on the specific methods of acm.Condition.

---

construct: function (field, cond)

Description: Constructor for the *acm.Condition*-class.
1) Sets default values for the element.
2) Creates UI objects.
3) If available, sets the data given to the elements.
Parameters:

- *field* **String** The text value of the field (one row).
- *cond* **String** The value of the condition that's going to be set as currently selected.

Return: ---

Ontonut.js (acm.Ontonut)
Extends: acm.Component

Description: acm.Ontonut is the object designed to store the component data of the UI. **Warning**, many methods in this class have been deprecated by recent changes on how data is retrieved from agent memory. The methods in this class, which are intended for parsing, can still be used, but caution is advised.

Following sections will contain information on the specific methods of acm.Ontonut.

---

construct: function (tokensArray, mapArray, onutID)

Description: Class constructor. If arrays are available, this method will parse the data in them.
Parameters:

- *tokensArray* **[Int]** An array containing the token numbers for the text map (*mapArray*). Both arrays length should be equal.
- *mapArray* **[String]** An array containing the (SAPL-format) text. Both arrays length should be equal.

Return: ---

QResultTable.js (acm.QResultTable)
Extends: qx.ui.window.Window

Description: This class is designed to create a window with call/query results. The window contains a plain table-element to which the call/query results will be added.

Following sections will contain information on the specific methods of acm.QResultTable.

construct: function (label, queryData)

Description: Class constructor.
1) Sets default values for this window.
2) Initiates data parse, which in turn will create the UI elements (table containing query results). 3)
Sets some event handlers.

Parameters:

• *label*	**String**	The label/caption of the window.
• *queryData*	**[String]**	An array of query results to be processed.

Return: ---

SAPLParser.js (acm.SAPLParser)
Extends: qx.core.object

Description: This class tokenizes the SAPL code (plain text) given, and constructs SAPL statements.

Following sections will contain information on the specific methods of acm.SAPLParser.

tokenize: function (c)

Description: tokenize-method divides the piece of code given into tokens and stores them. Each piece of text is given a number to symbolize its type. 1 = "normal text". 2 = "dot". 4 = "beginning of a container". 8 = "end of a container". 10 = "end of file".
Parameters:

• *c*	**String**	The piece of code to be tokenized.

Return: ---

parseBlock: function ()

Description: Once *tokenize* has stored the tokens into object memory, this method will compose the tokens/pieces into SAPL statements.
Parameters: ---

Return: ---

---

parse: function (code)

Description: This method will first tokenize the code given and then use *parseBlock* to make statements out of it. Using this method is not required, individual methods can be used by themselves.

Parameters:

- *code* **String** The piece of code to be tokenized and re-constructed.

Return:

- *String* Return value of this method is the tokenized and re-constructed piece of code that was given.

3.4 Future opportunities

The ACM tool has been developed using Metso industrial maintenance data sources as test samples. In general, the tool can be applied to other problem domains as well. The main development direction will continue towards extending component types supported and then the component process chains building.