UBIWARE Deliverable D2.3:

UBIWARE Platform Prototype v.2.0

May, 2009

Date	May 15, 2009
Document type	Report
Dissemination Level	UBIWARE project consortium
Contact Author	Vagan Terziyan
Co-Authors	Oleksiy Khriyenko, Sergiy Nikitin, Michal Nagy
Work component	WP1, WP2, WP5
Deliverable Code	D2.3
Deliverable Owner	IOG, JYU
Deliverable Status	Mandatory, Internal
Intellectual Property Rights	Unaffected

Table of Contents

Introduction

The UBIWARE project aims at a new generation middleware platform which will allow creation of self-managed complex industrial systems consisting of distributed, heterogeneous, shared and reusable components of different nature, e.g. smart machines and devices, sensors, actuators, RFIDs, web-services, software components and applications, humans, etc. The technologies, on which the project relies, are the Software Agents for management of complex systems, and the Semantic Web, for interoperability, including dynamic discovery, data integration, and inter-agent behavioral coordination.

Work in this project is divided into seven work packages which are running in parallel:

1. Core agent-based platform design
2. Managing Distributed Resource Histories
3. Security in UBIWARE
4. Self-Management and Configurability
5. Context-aware Smart Interfaces for Integrated Data
6. Middleware for Peer-to-Peer Discovery
7. Industrial cases and prototypes.

Work-packages 1 through 6 are research work packages; however, the research efforts are combined with agile software development processes. Prototypes of the UBIWARE platform, integrating the work in these 6 work packages at different levels of their readiness, are developed during each project year, as UBIWARE 1.0, UBIWARE 2.0 and UBIWARE 3.0.

UBIWARE deliverable D2.1 reported on the research results from work packages 1 through 6 (it was decided not to perform the work at the WP3 during the second project year due to limitation in resources). This deliverable, D2.3, presents the integrated development results from those work packages, i.e. the current state of the UBIWARE platform prototype. Naturally, during the development stage, solutions described previously in D2.1 have undergone some changes and improvements.

The deliverable D2.3 presents UBIWARE 2.0. This deliverable consists of software itself and an accompanying report. It was decided not to perform the work at the WP3 during the second project year due to limitation in resources and develop the most important research findings of the year. Therefore, D2.3 integrates results from WP1, WP2 and WP5.

UBIWARE Deliverable D2.3:
Workpackage WP1:
Task T2.2_w1:

1 UbiCore – Core Distributed AI platform design

1.1 Platform Changes

During the 2nd project year the core UBIWARE platform has become more mature. The platform performance has been significantly improved by introducing indices within the internal belief storage model. Also a number of bugs were fixed; some of them were critical, i.e. caused malfunctioning of scripts and behaviors. From the functional point of view the changes were minor, yet some of them bring additional important features.

1.1.1 UBIWARE platform as a service

Typically, a Ubiware-based application consists of several agents running within one or more JADE containers. The usual way of starting an application is to execute several scripts, one script per agent. This approach has two main disadvantages. Firstly, several scripts have to be run (each in separate window) which is not always possible if a number of agents grows. Secondly, if an agent crashes for some reason, it will not be restarted which will most likely make the application maintenance unreasonably complicated.

In order to overcome these limitations we decided to integrate Java Service Wrapper Community edition (http://wrapper.tanukisoftware.org/) to the Ubiware platform. It allows us to run a Ubiware-based application as a service. This has the following advantages:

- *The application is considered a service.* It is not needed to run several scripts. Also, if the application behaves as a Windows/Linux service, it may be started automatically every time the system is started.
- *The application is restarted every time it crashes.* There are several ways to define when and how the application should be restarted in case one of the agents dies.
- *The configuration file of the wrapper is platform independent.* This allows us to use one configuration file for several platforms. This basically means that if the application is run as a service on one platform (e.g. Windows) it can be run as a

service also on another platform (e.g. GNU/Linux) without the need to change the configuration file.

- *Java Service Wrapper uses GNU GPLv2 license.*

In order to run Ubiware application as a wrapper a configuration file *wrapper.conf* needs to be edited. The configuration file resides in the main directory of the Ubiware platform. The configuration file contains a set of properties. The most important property is *wrapper.app.parameter*. This parameter contains the information about the application being wrapped. In case of the Ubiware application, it should look like this:

```
wrapper.app.parameter.1=jade.Boot
wrapper.app.parameter.2=-detect-main
wrapper.app.parameter.3=false
wrapper.app.parameter.4=
Agent1:ubiware.core.UbiwareAgent(sourceCode1.sapl);
Agent2:ubiware.core.UbiwareAgent(sourceCode2.sapl);
```

where:

`Agent1` is the name of the first agent
`sourceCode1.sapl` is the path to the source code of the first agent
; is used for the separation of agent declarations

If you want to deploy the application as a Windows service, you can customize it with these parameters:

```
# Name of the service
wrapper.ntservice.name=UbiwareApp
# Display name of the service
wrapper.ntservice.displayname=UBIWARE Application XYZ
# Description of the service
wrapper.ntservice.description=UBIWARE Application
Description
# Mode in which the service is installed.  AUTO_START or
DEMAND_START
wrapper.ntservice.starttype=AUTO_START
# Allow the service to interact with the desktop.
wrapper.ntservice.interactive=false
```

After the configuration file is written, we may install the application as a service. This can be done by executing wrapper.exe (Windows) or wrapper (Linux) from the platform folder with a parameter –i:

```
wrapper.exe –i
```

or

```
wrapper –i
```

To uninstall the service, you have to execute wrapper.exe (Windows) or wrapper (Linux) with the parameter –r:

```
wrapper.exe –r
```

or

```
wrapper –r
```

1.2 S-APL language changes

The S-APL language as such has not undergone significant changes. The main construct that was added is a new operator for use in a new type of rules - an *Inference Rule*:

```
{{...} ==> {...}} sapl:is sapl:Rule
```

`==>` is the shorthand for `sapl:infers`. If a behavior rule is used for semantic inference (generating new facts from existing ones), one needs to:

1. add to the head of the rule the negation of the tail the rule (see “exclusive condition” below) – to avoid continuous non-stop execution of the rule;
2. use a set of sapl:All wrappings - for all relevant variables – to enforce that the rule infers all possible facts in one iteration.

When using `==>`, these two things are done automatically – negation of the tail is checked and the rule is executed for every solution found – the rest being exactly the same as for `=>`.

UBIWARE Deliverable D2.3:
Workpackage WP2:
Task T2.2_w2:

2 UbiBlog – Managing Distributed Resource Histories

In UBIWARE, every resource is represented by a software agent. Among major responsibilities of such an agent is monitoring the condition of the resource and the resource's interactions with other components of the system and humans. The beliefs storage of the agent will, therefore, naturally include the history of the resource, in a sense "blogged" by the agent. Obviously, the value of such a resource history is not limited to that particular resource. A resource may benefit from the information collected with respect to other resources of the same (or similar) type, e.g. in a situation which it faces for the first time while other may have faced that situation before. Also, mining the data collected and integrated from many resources may result in discovery of some knowledge important at the level of the whole ubiquitous computing system. A scalable solution requires mechanisms for inter-agent information sharing and data mining on integrated information which would allow keeping the resource histories distributed without need to copy those histories to a central repository.

During WP2's Year 1 (the *Sharing* phase), needed mechanisms were designed for effective and efficient sharing of information between different agents, e.g. representing different resources. S-APL was used as the communication content language, which has enabled:

- One agent to query another agent for some information, using the query constructs similar to that of SPARQL but with even wider range of possible filtering conditions
- One agent to inform another agent, i.e. to proactively push some information of any complexity.
- One agent to request another agent to perform some actions, either an atomic behavior or a complex plan involving a set of rules and atomic of complex behaviors.

During WP2's Year 2 (the *Integration* phase), we have been working on the following question:

How to realize the possibility of querying a set of distributed, autonomous, and, hence, inevitably semantically heterogeneous resource histories as they were one virtual database, i.e. how to collect and integrate needed pieces of information from distributed sources?

The research we have conducted during the second year has resulted in the theoretical basis for the distributed querying of heterogeneous resources. We have introduced Ontonuts technology – an approach that allows us to represent physically distributed data in one place and perform queries over this data as if it was integrated into one storage. The theoretical foundations of the technology can be found from the deliverable D2.1. In this report we will present an Ontonuts engine – a complex component that implements the above mentioned functionality.

2.1 Ontonuts engine

2.1.1 How it works (a reminder)

The Ontonut capabilities are S-APL descriptions with explicitly defined preconditions and effects.

$$Ontonut : \{script, precond, effect\} \quad (2.1)$$

The semantic annotation of Ontonut (by precondition and effect) allows us to automatically plan (compose) agent's activities to achieve a specified goal. The script part has an S-APL code that produces the effect based on the precondition.

The engine matches user-defined calls (three types of calls) against Ontonut descriptions and produces an execution plan in terms of normal S-APL commitments. The plan is then performed by the Agent's engine and the result is produced. There is also a special type of Ontonuts – called Donuts – specifically tailored to solve the database connectivity. The engine supports Donut descriptions and Donut calls are specifically handled by the engine, i.e. the database is queried not explicitly from the agent's beliefs, but rather from the Ontonuts engine.

2.1.2 Software implementation of the engine

The engine as a complex component consists of a set of S-APL scripts and Java-classes (see Figure 2.1).

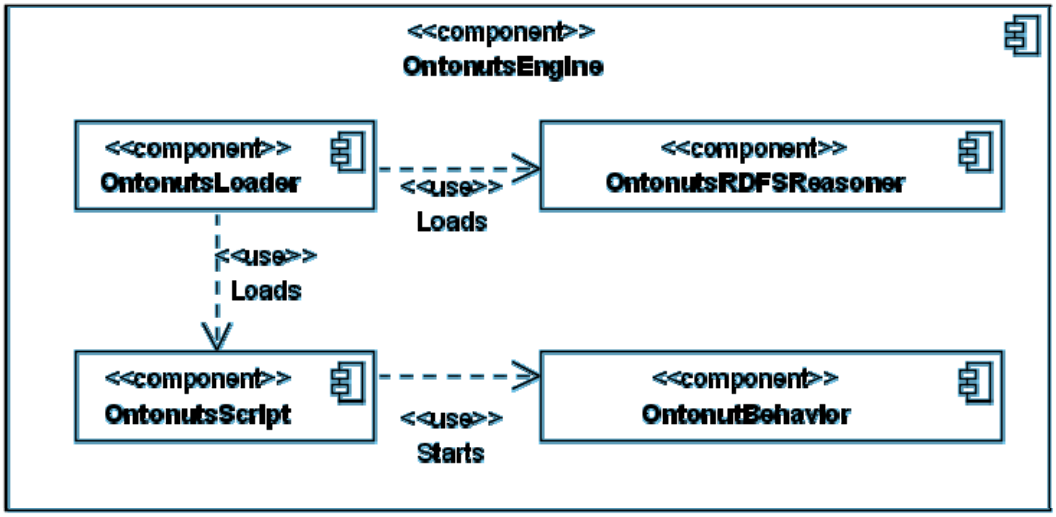

Figure 2.1 – Component view of the Ontonuts engine

The whole component initialization is done by the OntonutsLoader script. The script ensures the order of subcomponent loading and therefore guarantees proper Ontonuts engine initialization within the agent script. The sequence of actions is defined as follows:

1. Load Reasoner (OntonutsRDFSReasoner.sapl)
2. Load Ontonuts script (Ontonuts.sapl)
3. Load user defined Ontonut definitions
4. Load user defined main business logic

The engine provides two main access points (Figure 2.2):

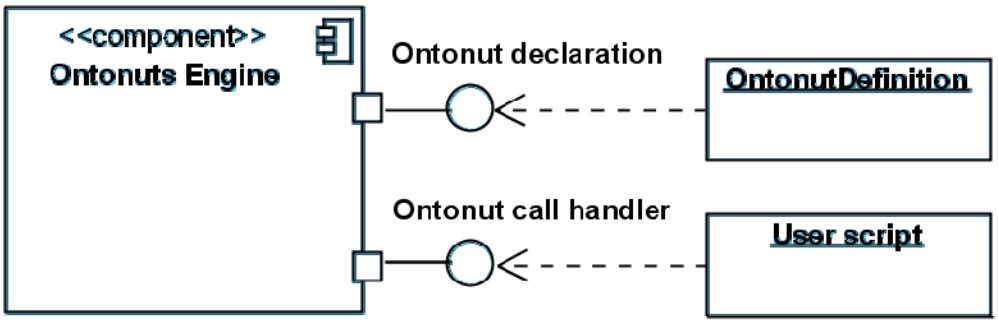

Figure 2.2 – External interfaces of the Ontonuts engine

2.1.3 Handling ontonut definitions

All the Ontonut definitions within the agent's beliefs (in G-container) are processed by the engine script. The definitions can be either active or passive. Passive definitions are not used for planning and execution, however they stay in the memory. Enabling and disabling of definitions is regulated via flag `di:isActive`. For example:

```
od:OntonutDef_1 di:isActive true.
```

The Ontonut definitions are separated from the business logic script because these are the reusable user-defined components that do not depend on the particular case-specific application.

When Ontonut definitions are loaded, the OntonutsReasoner immediately performs semantic reasoning over the preconditions and effects of all Ontonuts. The reasoner uses user-defined ontology that should be placed to agent's general context G.

2.1.4 Handling Ontonut calls

The main processing logic of the engine is performed in the OntonutBehavior. Ontonuts S-APL script is rather a wrapping interface between the Java-based RAB implementation of the OntonutBehavior and S-APL declarations and calls. The Ontonuts script is triggering the OntonutBehavior when an Ontonut call appears, thus helping to avoid useless RAB initializations on every agent cycle.

The script of the agent interacts with the Ontonuts engine not directly, but by posting a predefined statement structure. The Ontonuts engine supports three types of Ontonut calls:

- Explicit
- Goal based
- Pattern-based

The *Explicit call* to the Ontonut has following syntax:

```
{sapl:I sapl:do ont:Ontonutid}
```

```
sapl:configuredAs {
   x:precondition sapl:is {Input statements}
}.
```

The result of the call is added to the G – a general context.

The *Goal-based call* is initiated by adding the following goal definition to G:

```
sapl:I ont:haveGoal :id.
:id ont:goalDef{goal statements}
:id ont:initData {initial data}
```

The third type – *a pattern-based call* is triggered when the content of the active commitment in its left part matches the effect pattern of at least one Ontonut.

```
{A A ?a} => {some action with ?a }.
```

Let us consider goal-based call processing as a fundamental element of the goal-driven agent operation (pattern-based call can be considered as a subset of a goal-driven one). The sequence diagram of the goal-based call is shown on Figure 2.3.

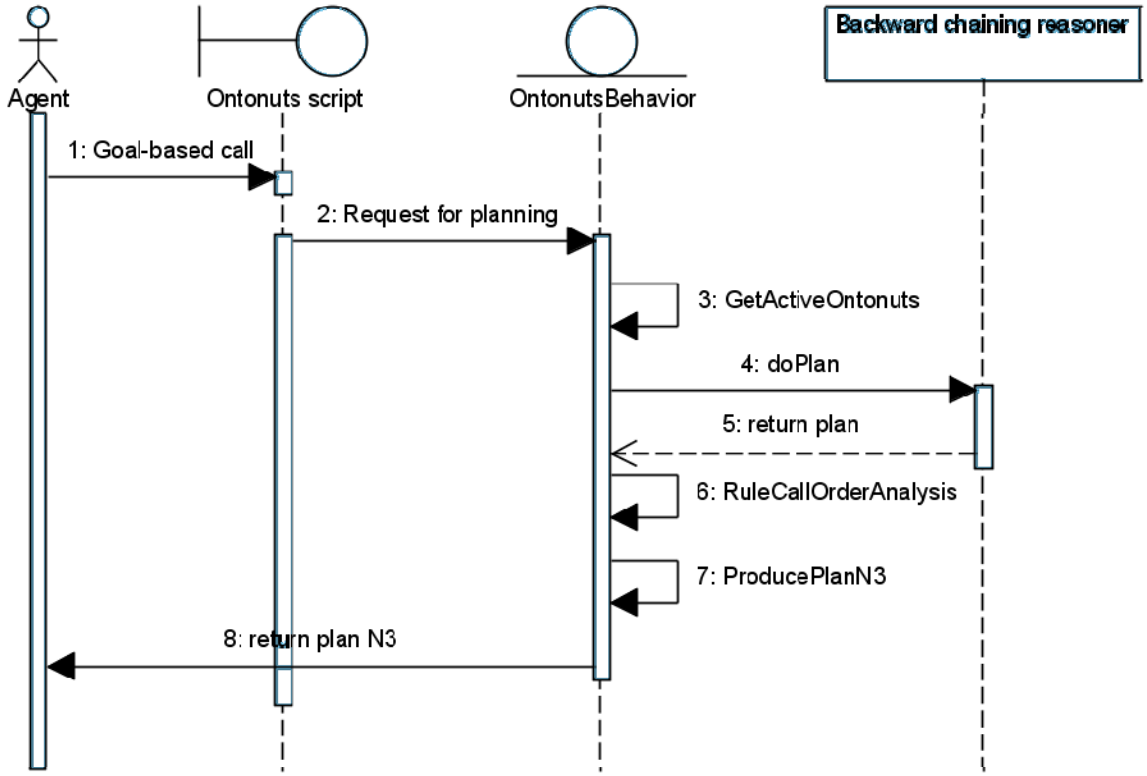

Figure 2.3 – Sequence diagram of the goal-based call

The OntonutBehavior receives a goal state (definition of a desired state) specified by the agent and performs complex multistage operation over the goal received and the data and capabilities (Ontonuts) available within agent's beliefs. As soon as the OntonutBehavior is initialized by the Ontonuts script, it takes all the active Ontonut descriptions and forms a request to the backward chaining reasoner to provide a plan of possible actions that lead to the goal. The reasoner is developed as an independent component and provides an interface for OntonutBehavior, therefore the request needs to be formulated properly. The reasoner sequentially (algorithm is presented in the research deliverable D2.1) builds a plan and returns it in an internal format. The plan is not yet ready to be put to the execution. In fact, the reasoner only gives the answer whether the goal is achievable or

not within the conditions specified and divides sets of required capabilities into several ordered layers. However, the execution order within those layers is not specified. The OntonutBehavior performs rule call order analysis for each layer and defines precise execution scheme. It is important to note here, that the engine takes into account intermediary results of the execution when it generates the plan. So, the plan being executed will keep the results of the previous steps and use them if required in subsequent steps. As soon as the execution order is set, OntonutBehavior serializes the plan into the N3 format which is put to the agent's beliefs for execution.

Similarly, the explicit Ontonut call to the engine goes via Ontonuts script to the OntonutBehavior. The engine detects the type of the Ontonut (in the example diagram on Figure 2.4 it is a Donut call) and handles the call accordingly.

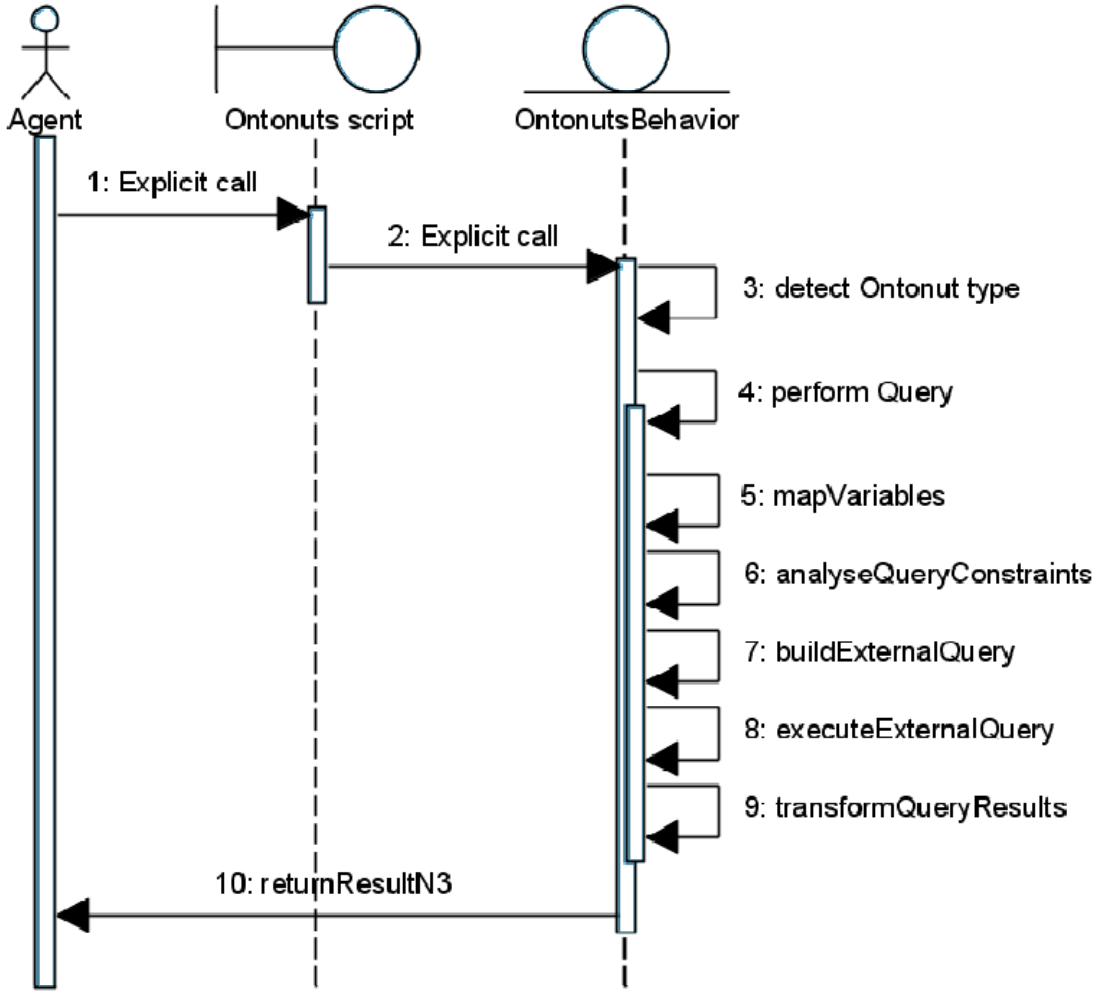

Figure 2.4 – An explicit Donut call to the engine

The Donut call in the example above is detected on the 3rd step and the 4th step involves 5 sub-steps. First of all, the variable mappings are very essential part in the engine operation. The engine allows the user to specify variables for Ontonut definitions as local ones and use different variable names independently in the business logic. The matching and mapping of variables is performed "on-the-fly" by the engine. Even more, the variables used in different Ontonut definitions, when composed to one execution plan, still have no naming limitations. The mapping and initialization of variables is handled by the engine, therefore Ontonut descriptions can be developed independently from each other.

The analysis of the constraints (step 6 on the diagram) is a very important step in sequential execution of queries, especially when results of preceding sub-query are used in subsequent sub-query generation. This feature allows the engine to limit significantly the amount of data being queried remotely and in some cases it is the only possible approach for distributed query execution, because some logging systems may contain huge amount of records even in one table, which as a whole, when extracted may not fit the physical memory of the agent platform. Therefore certain queries must have constraints. Here the engine not only decomposes query to sub-queries, but also automatically specifies dependencies and order among them and therefore the sub-queries are not executed independently, but use the results of the preceding sub-queries to constrain the range of dependent variables in subsequent queries (step 7 on the diagram). As soon as (sub-) query is executed (step 8), the results of the query are transformed from the native data model to the ontology-based one (step 9). The results are then added to the current execution container of the plan.

2.1.5 Organizing the plan execution

Each plan is supplied with the execution container – a temporary storage place for intermediate results of execution. As soon as goal is achieved, the result is taken from the execution container to the resulting container. The generated execution plan also contains status markers for each sequential step. Each step is executed only upon appearance of marker about successful execution of the previous step. The execution step can also be marked as a failed one. In this case the current plan execution is stopped and the whole plan is marked as failed.

The figure 2.5 below shows simplified agent belief structure when a goal-based call takes place. The Ontonut (Donut) definitions are representing the databases, therefore their preconditions are always true (you do not need any pre-requirements to query the database, it is already accessible). The effects of the Ontonuts are representing the query patterns that can be answered. To keep it simple, we have represented the database content in an adapted triple format. In the reality, however, we need one more transformation and mapping layer in between the agent beliefs and the external data. Nevertheless, the absence of the layer does not affect the functional side of the example, as we are explaining the plan execution steps here. Next, after the Ontonut definitions, you may find the goal definition, which looks pretty much as a query. So the goal defines the question: Is there any record set, such that satisfies the following condition:

C C ?a and *B B ?a*, where *?a* is less than 10

The initial data specified with the goal has a default value – *precondition is alwaystrue*. This definition is used in all standard Donut preconditions. When the plan is produced, a temporary execution container is initialized: *:plan1 di:execContainer {}*. The container is the place for temporary query and execution results. At the same time the first step of the plan is started by the explicit Ontonut call to the *O1*. The call defines a query sub-graph – a triple pattern to be answered. Next there is a filtering statement put in a separate container. It will be used by the engine to produce native query to the database. The call defines two control actions: the statements to be added in case of successful execution and in case of failure. In the particular case, the statement *:plan1 di:execStep2 di:start* is added upon successful execution of the O1 call. This statement is defined as a precondition for the next step, where a call to the O2 ontonut is performed. After the execution of the explicit call to the Ontonut O1, we have the result in the temporary

container (as shown on Figure 2.5, under the middle yellow line). When the second Ontonut call is run, one may notice the change in input parameters – a statement *di:varValuesPattern sapl:is {B B ?a}* is added, whereas the filtering statement is not specified anymore. The reason for such a change is that filtering condition (*?a < 10*) has already been satisfied and there is no need for it within the second call, because filtering will be done using the output of the previous call. Therefore the engine uses the pattern to query the temporary execution container and get values of the variable *?a*. The values are then used to build a query to the DB2. When the second query is performed, the final result is put to the *:plan1 di:result {Result container}* (see Figure 2.5).

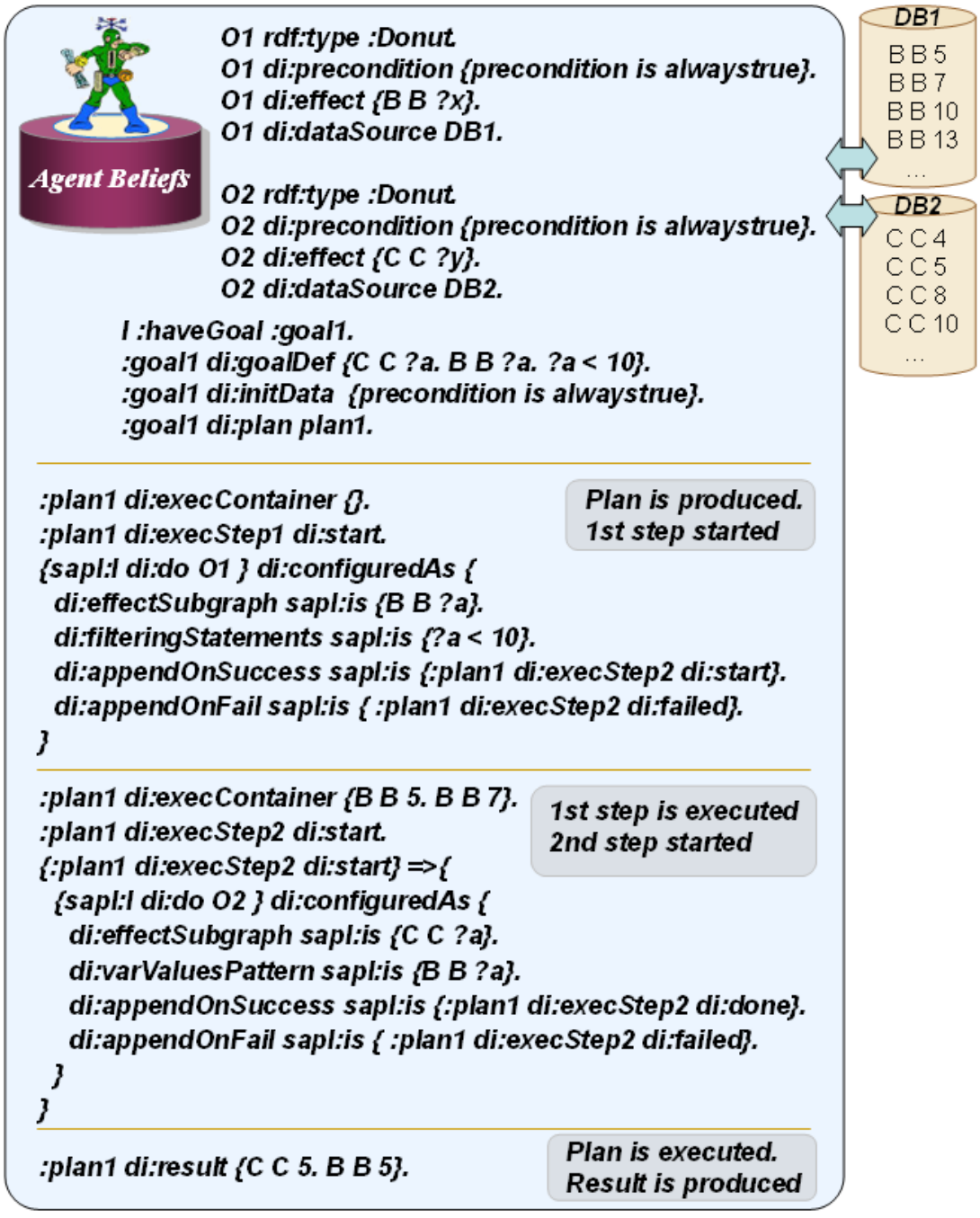

Figure 2.5 – Ontonut engine planning and execution example

2.1.6 Implementation of the OntonutBehavior

The OntonutBehavior is the "brain" of the Ontonuts engine. It is fully implemented in Java. Below is the UML diagram of the package classes that were developed (Figure 2.6).

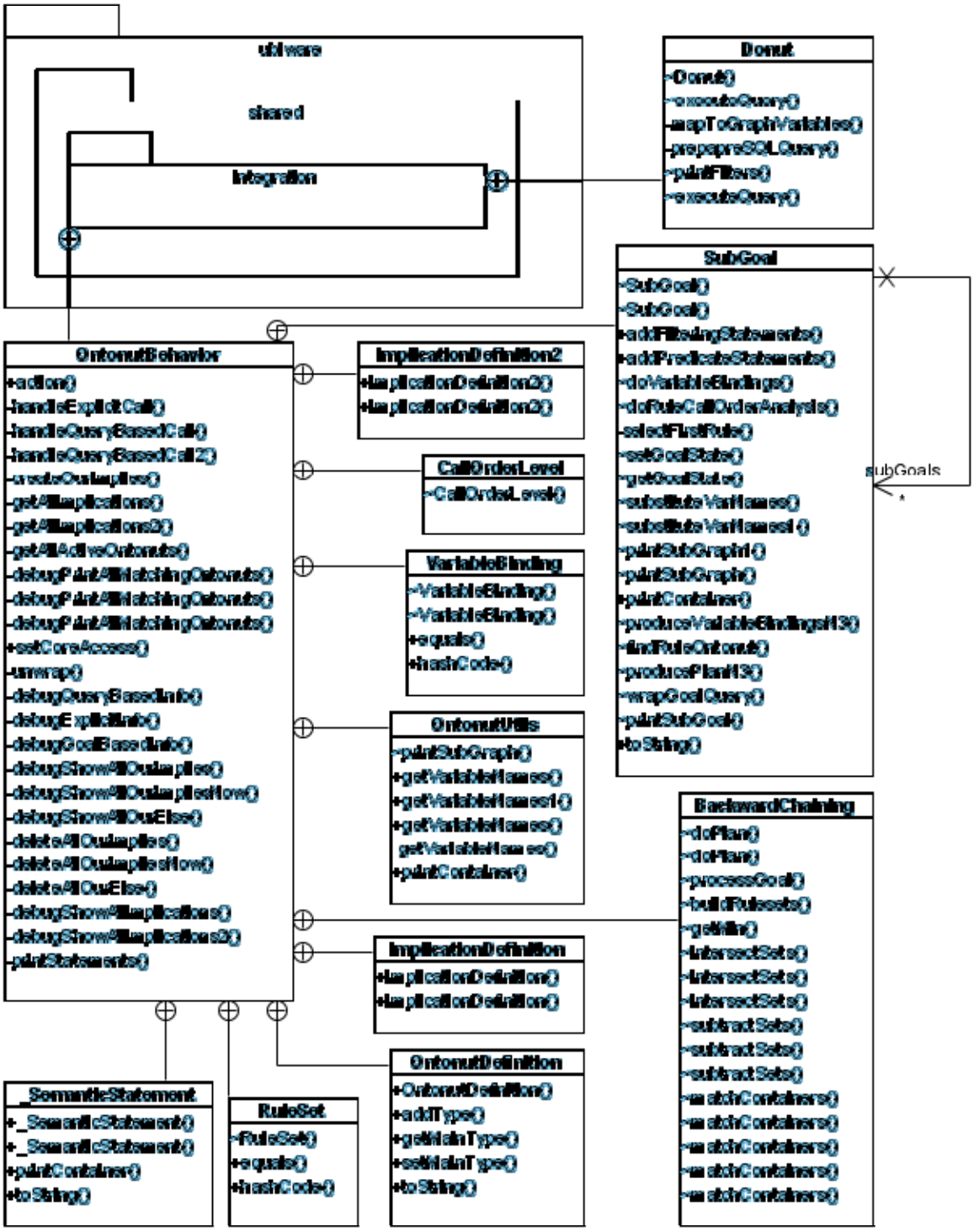

Figure 2.6 – UML diagram of Java classes

The main logic of the engine spreads over OntonutBehavior, SubGoal and BackwardChaining classes. The features of the database connectivity are encapsulated in the Donut class. The OntonutBehavior class serves as a dispatcher component, i.e. it identifies the types of Ontonut calls and takes respective actions.

2.1.7 Donuts support in the engine

As we have described in the research deliverable D2.1, the Ontonuts engine provides simple functionality for the user to query database sources. The same functionality is utilized by the engine itself, when it generates plans involving database querying. This functionality in fact is an explicit Ontonut call, same as others, from the user's point of view. However, for the engine, the type of the Ontonut being invoked differs. The special support for a subclass of Ontonuts called Donuts was implemented taking into account the theory of General Adaptation Framework (Kaykova et al.). The Donut class is the sub-component of the engine that implements functions of query preparation, transformation, execution, etc. The entry point of the Donut is the class constructor, where all the database parameters are initialized, such as URL of the database, username, password and the SQL query base. The SQL Query base should be understood as:

SQL query base – is a generic SQL construct that extracts all the data needed for provision of all Donut instances from the resource it connects to.

At the next step, the Donut provides a function *prepareSQLQuery()* for generation of the full SQL query taking into account filtering constructs and results of previous queries. This step also includes the transformation of the query from the internal agent ontology (in S-APL) to the native database model. The transformation function uses the mapping definitions of the Donut. In the example above (Figure 2.7) one can see the dependency between the SQL query base, the effect definition and the mapping definition of the Donut:

```
od:DNADiaryEntry di:queryBase .
 "SELECT EntryID, EntryDate, Author, Title, Description FROM dbo.Entry ".
od:DNADiaryEntry di:mapping {.
  ?entryid di:mapsTo "EntryID"..
  ?date di:mapsTo "EntryDate"..
  ?author di:mapsTo "Author"..
  ?title di:mapsTo "Title"..
  ?rowid di:mapsTo {10 sapl:is "<"..
                    11 sapl:is od:DNADiaryEntry..
                    12 sapl:is "#"..
                    13 sapl:is ?entryid..
                    14 sapl:is ">".
                    }..
  ?text di:mapsTo "Description"..
}..
od:DNADiaryEntry di:precondition {di:precondition sapl:is di:alwaystrue}.
od:DNADiaryEntry di:effect {.
          ?rowid rdf:type od:DNADiaryEntry..
          ?rowid :entryId ?entryid..
          ?rowid :entryDate ?date..
          ?rowid :entryAuthor ?author..
          ?rowid :entryTitle ?title..
          ?rowid :entryText ?text..
}..
```

Figure 2.7 – Example of mapping definition of the Donut

The mapping definition establishes the correspondence between variable names of the effect pattern and names of the SQL query base variables. The definition allows for more complex specifications, where an effect variable is mapped to a concatenation of strings, resource URIs and other effect variables. Fro example, given that *?entryid* variable value equals 100, then *?rowid* variable is produced as:

"<" + "od:DNADiaryEntry" + "#" + "100" + ">"

and gives: *<od:DNADiaryEntry#100>*

2.2 Summary

The development of the Ontonuts engine presented here was heavily inspired by the industrial domain problems, in particular by the heterogeneity of information systems within the same organization. We have prioritized the development of parts of the engine to serve concrete needs of distributed querying within real infrastructure. The further growth of the engine will be directed towards the improvement of three main directions:

1. Connectivity (extend engine to support more data source types)
2. Componentization (design generic mechanisms for component execution management)
3. Planning (extend planning capabilities of the engine by introducing utility and multi-choice plans for the engine)

The extensions described above allow us to merge the problems of WP1 and WP2 for the next year R&D activities because the approach and the mechanisms to solve the tasks specified in WP1 and WP2 have common ground and thus can be naturally integrated.

UBIWARE Deliverable D2.3:
Workpackage WP5:
Task T2.2_w5:

3 Smart Interfaces: Context-aware GUI for Integrated Data (4i technology)

This workpackage studies dynamic context-aware Agent-to-Human interaction in UBIWARE, and elaborates on a technology which we refer to as 4i (FOR EYE technology). From the UBIWARE point of view, a human interface is just a special case of a resource adapter. We believe, however, that it is unreasonable to embed all the data acquisition, filtering and visualization logic into such an adapter. Instead, external services and application should be effectively utilized. Therefore, the intelligence of a smart interface will be a result of collaboration of multiple agents: the human's agent, the agents representing resources of interest (those to be monitored or/and controlled), and the agents of various visualization services. This approach makes human interfaces different from other resource adapters and indicates a need for devoted research. 4i technology will enable creation of such smart human interfaces through flexible collaboration of an Intelligent GUI Shell, various visualization modules, which we refer to as MetaProvider-services, and the resources of interest.

WP5's Year 2 elaborates on probably the most important part of 4i vision, which can be called "context provision". Especially when considering a human, presenting information on a resource of interest alone is not sufficient - information on some "neighboring" objects should be included as well, which form the *context* of the resource. The ability to determine what type of context in right one for the situation and collecting the information that forms the context of that type for a specific resource is central in 4i vision.

3.1 Background and previous development

One of the 4I Browser enhancements concerns a smart and intelligent technique for automatic dynamic selection of a visualization context. The logic is based on a history of visualization contexts and resources that users have browsed/visualized previously. This context ranking technique allows us to sort a list of visualization contexts in more appropriate order for user and give him/her a hint for next logical step in though resource

browsing process. Thus, it can become a smart search system that leads the user in proper direction/way.

To make the browsing process more user-friendly and reduce amount of useless manipulations from the user side, browser automatically visualized currently selected resource in a context that is considered as a default context for the class of selected resource. Such feature makes browser more dynamic and handy for the user, but can be useless sometimes and imposes unnecessary default visualization on user. Keeping going to the direction of context selection enhancement, we decided to enhance the browser with context ranking functionality. This add-on rank (sort by relevance to the current situation) the list of visualization contexts for user depending on the user browsing history (current browsing route - a sequence of visualized resources and correspondent visualization contexts) and the experience of other users (history of browsing routes).

The idea is to keep all the user browsing routes in database and compare browsing route of the current user with them. So, the more similar the current route to some routes from the history database, the higher probability that a visualization context (chosen by predecessors) for current resource fits the needs of current user. The result of such add-on is a list of sorted visualization contexts in a context of browsing history and experience of predecessors (see Figure 3.1). The most relevant context should be applied for visualization of recently chosen resource. If there is no matching with the previous browsing routes in the history database, then the default context will be chosen (as has been implemented in the previous version of the browser).

History database (HistoryDB) has a table that contains the browsing routes and correspondent contexts that has been chosen for visualization of the last resources in the routes. Browsing routes is presented as a sequence of pairs "**Resource_ID:Context_ID**" that show in which context certain resource has been visualized. The structure of the table is:

- **id** unique identifier of the record;
- **resID** unique resource identifier (last resource in the sequence of browsing route);
- **route** tail of the browsing route;
- **routeN** amount of the same browsing routes;
- **usedContextID** unique identifier of a context that has been used for visualization of subject resource (*resID*);

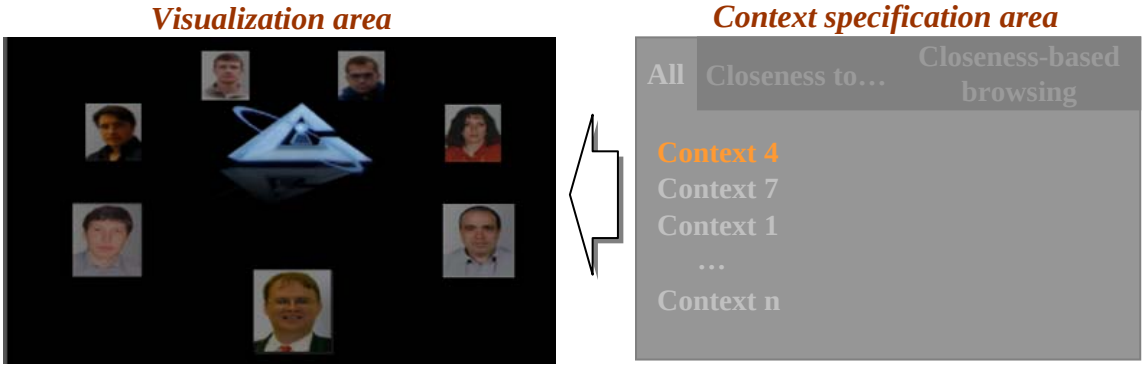

Figure 3.1 – Context-based resource visualization.

This functionality has been developed during the first part of the project year and presented during the Deliverable 2.1 as a status report.

There are a lot of contexts in which resources may be visualized. But, very often user faces a need to find similar/close resources to the initial one. Thus, we decided to include a visualization of the resources in a context of their similarity/closeness to the 4i(FOR EYE) Browser as its an inherent functionality.

One of the industrial cases that has been developed during the second project year and delivered in Deliverable 2.2 was focused on Idea similarity visualization and browsing. The main development in the direction of Deliverable 2.3 has been done during that period (see Deliverable 2.2):

- Distance Measuring Methods for five main resource description types and General Resource Distance Measuring Method;
- format and schema of similarity context related data;
- resource closeness/similarity Visualization component.

During the period after the previous checkpoint we add useful functionality to the system. Current implementation of the 4I GUI Shell supports visual configuration of resource similarity visualization context.

3.2 Visual configuration of resource similarity/closeness visualization context

Additionally to the previous development we added possibility to create new, delete and modify the similarity contexts. Such visualization context implies user specification of the resource properties significance and existence of additional contextual information for the resources properties (depending on their types).

Comparison between the resources is performed based on common properties. Current implementation supports five types of such parameters (properties):

Text field types:

Type 1: Just a pure word/sentence. Additional contextual information for this field is its significance.

```
<fieldContext>
 <field_type>textField</field_type>
 <field_property_id>…</field_property_id>
 <field_property_name>…</field_property_name>
 <field_significance>…</field_significance>
 <field_calculation_method>…</field_calculation_method>
 <field_calculation_methods>
  <value>…</value>
   …
 </field_calculation_methods>
</fieldContext>
```

Type 2: Text field is presented by list of key words/sentences. Additional contextual information for this field is its significance.

```
<fieldContext>
 <field_type>keyWordsField</field_type>
 <field_property_id>…</field_property_id>
 <field_property_name>…</field_property_name>
 <field_significance>…</field_significance>
 <field_calculation_method>…</field_calculation_method>
 <field_calculation_methods>
  <value>…</value>
   …
 </field_calculation_methods>
</fieldContext>
```

Type 3: Text field is divided to the set of attributes and presented by correspondent list of values (words/sentences) of the attributes. In this case, the number of the attributes for certain text field should be defined and lists of possible (defined) values of the attributes should be defined and presented. In another words, it is defined amount of keywords, where each keyword is selected from a correspondent defined set of values. Additional contextual information for this field is the sets of values for each attribute (keyword) and the significance of the attributes, and as for all fields, significance of the field itself.

```
<fieldContext>
 <field_type>complexTextField</field_type>
 <field_property_id>…</field_property_id>
 <field_property_name>…</field_property_name>
 <field_significance>…</field_significance>
 <subprop_names>
  <subprop_name>…</subprop_name>
  …
 </subprop_names>
 <corClasses>
  <corClass>
   <class_significance>…</class_significance>
   <value>…</value>
   …
  </corClass>
  <corClass>
   <class_significance>…</class_significance>
   <value>…</value>
   …
  </corClass>
  …
 </corClasses>
 <field_calculation_method>…</field_calculation_method>
  <field_calculation_methods>
   <value>…</value>
    …
 </field_calculation_methods>
</fieldContext>
```

Number field: Just number that further will be normalized and compared. Additional contextual information for this field is its significance.

```
<fieldContext>
 <field_type>numberField</field_type>
 <field_property_id>…</field_property_id>
 <field_property_name>…</field_property_name>
 <field_significance>…</field_significance>
 <field_calculation_method>…</field_calculation_method>
 <field_calculation_methods>
  <value>…</value>
   …
 </field_calculation_methods>
</fieldContext>
```

Interval field: Field presented by start and end point on a numerical axis. Distance measuring function for such interval field is based on a distance between the centers of the intervals and the lengths of them. Additional contextual information for this field is the significance of these two main parameters, and as for all fields, significance of the field itself.

```
<fieldContext>
 <field_type>intervalField</field_type>
 <field_property_id>…</field_property_id>
 <field_property_name>…</field_property_name>
 <field_significance>…</field_significance>
 <field_calculation_method>…</field_calculation_method>
 <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>…</value>
  <value>…</value>
 </subField_significances>
 <field_calculation_method>…</field_calculation_method>
 <field_calculation_methods>
  <value>…</value>
   …
 </field_calculation_methods>
</fieldContext>
```

As we can see from these field type's descriptions configuration of the resource similarity context is specification of significances of resource properties/fields, subfields (in case of *complexTextField* and *intervalField*) and distance calculation method if there are several of them (for the moment we have three calculation methods for *intervalField* and three methods for the general resource closeness/similarity measurement).

Figure (Figure 3.2) shows us an interface for such context configuration/personalization that user may call by clicking the "Edit/Create" button located under the list of visualization contexts on the main window of 4I Browser. Here user has the access to the context parameters that can be changed and has a possibility either to edit currently selected context or to create a new one (in this case the name of a new context should be specified in the field next to the "Save" button) by pressing the "Save" button.

The distance calculation methods utilize coefficients that have the values in diapason [0..1]. To simplify the interface and make it more user friendly, we decided to consider the ***"absolute significance"*** of the resource fields as percentages from the full influence of the fields. In this case the sum of the fields' significances should be equal 100%. The same approach has been applied for the sub fields if there are any. For the "absolute significance" system supports two modes:

- ***fully user controlled mode:*** In this mode user can set any level of significance he/she wants. But at the same time, he/she takes a responsibility to check the condition that sum of the correspondent fields equals 100. To enter this mode user should uncheck the "recalculation" checkbox.

- ***mode with automatic recalculation of the significances:*** This mode will be useful in case when user change just the value of one field and do not want to change the values of other fields to fit the necessary conditions. In this mode system automatically recalculate the values of other fields proportionally to the previous values.

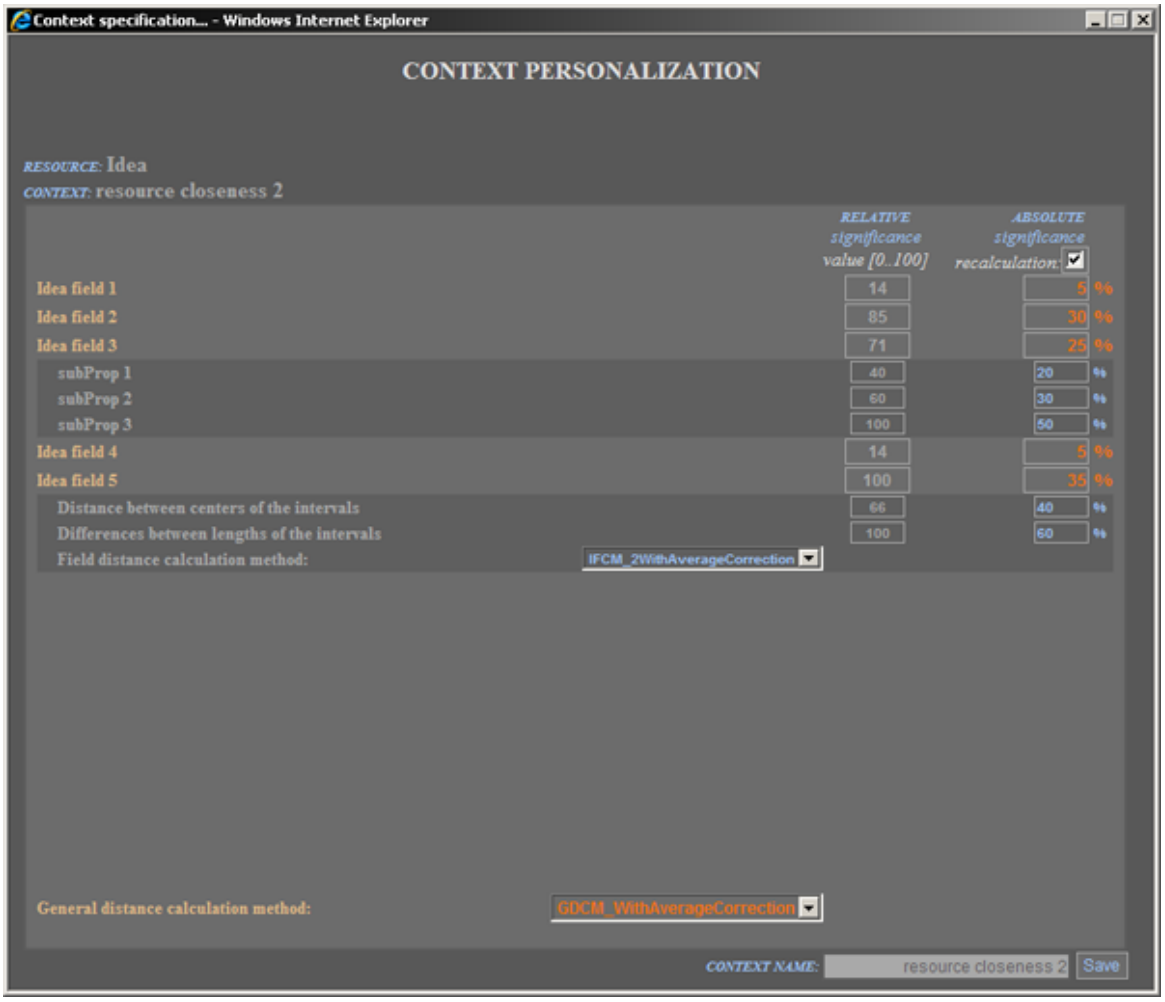

Figure 3.2 – Context configuration/personalization.

Sometimes it becomes difficult to define the significance for all the fields in percentages, and user prefers to specify ***"relative significance"*** for the field/property. In this case user estimates the significance of each field/property by value from 0 to 100 separately. With the "relative significance" the absolute values do not make sense, only comparative

differences if the values are taken into account. Further system itself transforms these values to the “absolute significance” and user can play with percentages later on if he/she wishes.

As ever user saved the changes or created a new resource similarity context, MetaProvider that provides visualization module creates and sends a new visualization to the 4I Browser accordingly to the new configuration.

3.3 Achieved results

One of the 4I Browser enhancements concerns a smart and intelligent technique for automatic dynamic selection of a visualization context. This context ranking technique allows us to sort a list of visualization contexts in more appropriate order for user and give him/her a hint for next logical step in though resource browsing process. Thus, it can become a smart search system that leads the user in proper direction/way.

Also we decided to include a visualization of the resources in a context of their similarity/closeness to the 4i(FOR EYE) Browser as its an inherent functionality. The main development in this direction has been done during the industrial case development period (see Deliverable 2.2):

- Distance Measuring Methods for five main resource description types and General Resource Distance Measuring Method;
- format and schema of similarity context related data;
- resource closeness/similarity Visualization component.

During the period after the previous checkpoint we add useful functionality to the system. Now current implementation of the 4I GUI Shell supports visual configuration of resource similarity visualization context.

Bibliography

Kaykova, O., Khriyenko, O., Kovtun, D., Naumenko, A., Terziyan, V., Zharko, A., (2005). General Adaption Framework: Enabling Interoperability for Industrial Web Resources, In: *International Journal on Semantic Web and Information Systems*, Idea Group, ISSN: 1552-6283, Vol. 1, No. 3, July-September 2005, pp.31-63.

Appendix A: UBIWARE Publications List (up to the April of year 2009)*

[1] Katasonov A., Terziyan V., Using Semantic Technology to Enable Behavioural Coordination of Heterogeneous Systems, In: V. Kordic (ed.), Semantic Web, IN-TECH Publishing, 2009, ISBN: 978-953-7619-33-6, 22 pp. (Book Chapter, to appear).

[2] Nagy M., Katasonov A., Khriyenko O. Nikitin S., Szydlowski M., Terziyan V., Challenges of Middleware for the Internet of Things, In: A. Lazinica (ed.), Robotics, Automation and Control, IN-TECH Publishing, 2009, ISBN: 978-953-7619-39-8, 24 pp. (Book Chapter, to appear).

[3] Kesäniemi J., Katasonov A., Terziyan V., An Observation Framework for Multi-Agent Systems, In: Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009), April 21-25, 2009, Valencia, Spain, IEEE CS Press, 6 pp.

[4] Katasonov A., Terziyan V., Semantic Approach to Dynamic Coordination in Autonomous Systems, In: Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009), April 21-25, 2009, Valencia, Spain, IEEE CS Press, 9 pp.

[5] Terziyan V., Zhovtobryukh D., Katasonov A., Proactive Future Internet: Smart Semantic Middleware for Overlay Architecture, In: Proceedings of the Fifth International Conference on Networking and Services (ICNS-2009), April 21-25, 2009, Valencia, Spain, IEEE CS Press, 6 pp.

[6] Nikitin S., Katasonov A., Terziyan V., Ontonuts: Reusable Semantic Components for Multi-Agent Systems, In: Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009), April 21-25, 2009, Valencia, Spain, IEEE CS Press, 8 pp.

[7] Khriyenko O., Adaptive Semantic Web based Environment for Web Resources, In: Jyvaskyla Studies in Computing, PhD Thesis, Volume 97, Jyvaskyla University Printing House, 192 pp., December 13, 2008.

[8] Bleier A., A Framework for Market-Based Coordination in Multi-Agent Systems, MSc Thesis, University of Osnabrück, September 30, 2008.

[9] Terziyan V., Semantic Web Services for Smart Devices Based on Mobile Agents, In: D. Taniar (Ed.), Mobile Computing: Concepts, Methodologies, Tools, and Applications (6 volumes), IGI Global, November 2008, ISBN: 978-1-60566-054-7, Vol. II, Chapter 2.22, pp. 630-641.

[10] Terziyan V. and Katasonov A. (2008) Global Understanding Environment: Applying Semantic and Agent Technologies to Industrial Automation, In: Lytras, M. and Ordonez De Pablos, P. (eds) Emerging Topics and Technologies in Information Systems, IGI Global , 2009, ISBN: 978-1-60566-222-0, pp. 55-87 (Chapter III).

[11] Katasonov A. and Terziyan V. (2008) Semantic Agent Programming Language (S-APL): A Middleware Platform for the Semantic Web, In: Proc. 2nd IEEE Conference on Semantic Computing, August 4-7, 2008, Santa Clara, CA, USA, pp.504-511.

* Papers are downloadable from http://www.cs.jyu.fi/ai/OntoGroup/UBIWARE_details.htm

[12] Khriyenko O., Context-sensitive Visual Resource Browser, In: Proceedings of the IADIS International Conference on Computer Graphics and Visualization (CGV-2008), Amsterdam, The Netherlands, 24-26 July 2008.

[13] Katasonov A., Kaykova O., Khriyenko O., Nikitin S., Terziyan V., Smart Semantic Middleware for the Internet of Things, In: Proceedings of the 5-th International Conference on Informatics in Control, Automation and Robotics, 11-15 May, 2008, Funchal, Madeira, Portugal, ISBN: 978-989-8111-30-2, Volume ICSO, pp. 169-178.

[14] Terziyan V., SmartResource - Proactive Self-Maintained Resources in Semantic Web: Lessons learned, In: International Journal of Smart Home, Special Issue on Future Generation Smart Space, Vol.2, No. 2, April 2008, SERSC Publisher, ISSN: 1975-4094, pp. 33-57.

[15] Katasonov A. and Terziyan V. (2007) SmartResource Platform and Semantic Agent Programming Language (S-APL), In: Proceedings of the 5th Conference on Multi-Agent Technologies (MATES'07), September 24-26, 2007, Leipzig, Germany, LNAI 4687, pp.25-36.

[16] Terziyan V., Predictive and Contextual Feature Separation for Bayesian Metanetworks, In: B. Apolloni et al. (Eds.), Proceedings of KES-2007 / WIRN-2007, Vietri sul Mare, Italy, September 12-14, Vol. III, Springer, LNAI 4694, 2007, pp. 634–644.

[17] Khriyenko O., Context-sensitive Multidimensional Resource Visualization, In: Proceedings of the 7th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2007), Palma de Mallorca, Spain, 29-31 August 2007.

[18] Khriyenko O., 4I (FOR EYE) Multimedia: Intelligent semantically enhanced and context-aware multimedia browsing, In: Proceedings of the International Conference on Signal Processing and Multimedia Applications (SIGMAP-2007), Barcelona, Spain, 28-31 July 2007.

[19] Khriyenko O., 4I (FOR EYE) Technology: Intelligent Interface for Integrated Information, In: Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS-2007), Funchal, Madeira – Portugal, 12-16 June 2007.

[20] Salmenjoki K., Tsaruk Y., Terziyan V., Viitala M., Agent-Based Approach for Electricity Distribution Systems, In: Proceedings of the 9-th International Conference on Enterprise Information Systems, 12-16, June 2007, Funchal, Madeira, Portugal, ISBN: 978-972-8865-89-4, pp. 382-389.

[21] Nikitin S., Terziyan V., Pyotsia J., Data Integration Solution for Paper Industry - A Semantic Storing, Browsing and Annotation Mechanism for Online Fault Data, In: Proceedings of the 4th International Conference on Informatics in Control, Automation and Robotics (ICINCO), May 9-12, 2007, Angers, France, INSTICC Press, ISBN: 978-972-8865-87-0, pp. 191-194.

[22] Naumenko, A., Srirama, S., Secure Communication and Access Control for Mobile Web Service Provisioning, In: Proceedings of International Conference on Security of Information and Networks (SIN2007), 8-10th May, 2007.

[23] Naumenko A., Semantics-Based Access Control in Business Networks, In: Jyvaskyla Studies in Computing, PhD Thesis, Volume 78, Jyvaskyla University Printing House, 215 pp., June 28, 2007.

[24] Naumenko A., Katasonov A., Terziyan V., A Security Framework for Smart Ubiquitous Industrial Resources, In: R. Gonzalves, J.P. Muller, K. Mertins and M. Zelm (Eds.), In: Enterprise Interoperability II: New challenges and Approaches, Proceedings of the 3rd International Conference on Interoperability for Enterprise Software and Applications (IESA-07), March 28-30, 2007, Madeira Island, Portugal, Springer, pp. 183-194.

[25] Naumenko A., SEMANTICS-BASED ACCESS CONTROL - Ontologies and Feasibility Study of Policy Enforcement Function, In: Proceedings of the 3rd International Conference on Web Information Systems and Technologies (WEBIST-07), Barcelona, Spain - March 3-6, 2007, Volume Internet Technologies, INSTICC Press, pp. 150-155.

Submitted or going to be submitted:

[26] Khriyenko O., Terziyan V., Similarity/Closeness-Based Resource Browser, In: Proceedings of the Ninth IASTED International Conference on Visualization, Imaging and Image Processing (VIIP-2009), July 13-15, 2009, Cambridge, UK, 7 pp. (submitted).