UBIWARE Deliverable D3.1:

Multi-Resource Orchestration in UBIWARE

October, 2009

Date	October 27, 2009
Document type	Report
Dissemination Level	UBIWARE project consortium
Contact Author	Vagan Terziyan
Co-Authors	Olena Kaykova, Oleksiy Khriyenko, Michal Nagy, Sergiy Nikitin
Work component	WP1&WP2, WP4, WP5
Deliverable Code	D3.1
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 D3.1 reports on the research results from work packages WP1&WP2, WP4 and WP5 (it was decided not to perform the work at the WP3 and WP6 during the third project year due to reduced resources).

The further text describes new developments on scientific concepts and results of the third project year, which are based on previously defined concepts and results obtained during the runtime of the UBIWARE project and which can be seen through attached UBIWARE list of publications (see Appendix A).

UBIWARE Deliverable D3.1:
Workpackage WP1:
Task T3.1_w1:
Workpackage leader: *Sergiy Nikitin*

1 UbiCore – Core Distributed AI platform design

During WP1's Year 3 (the *Coordination* phase), we will investigate the approaches where behavioral S-APL models are used not only as prescriptive tool (i.e. loaded by agent to act based on them), but also as descriptive tool - accessed by other agents to e.g. understand what to expect from or how to interact with the agent in question. WP1 will attempt to answer the following research questions:

- How to enable agents to flexibly discover each other, based both on the roles played and on particular capabilities possessed.
- What would be concrete benefits of and what mechanisms are needed for accessing and using a role's script by agents who are not playing that role but wish to coordinate or interact with an agent that does?

1.1 A background

The Ontonuts concept described in the research deliverable of the second project year (Bleier et. al., 2008) provides descriptive mechanisms for agent components. The true dynamism can be reached, when these components can be advertised to other agents (e.g. in a form of services). Furthermore, agents should be able to search for advertised service components and negotiate with other agents about conditions of service consumption. Such functionality would allow platform agents to plan, run and dynamically manage their processes with higher degree of freedom.

Converging with relevant technologies

Componentization and servicing are an intrinsic part of process management domain that involves very broad range of topics and perspectives. In this work we approach it from the software perspective and, in particular, business process planning and execution. The mainstream software approach towards business processes is service-oriented orchestration of process components. We align our work with the well known stack of web services[1] (Figure 1.1).

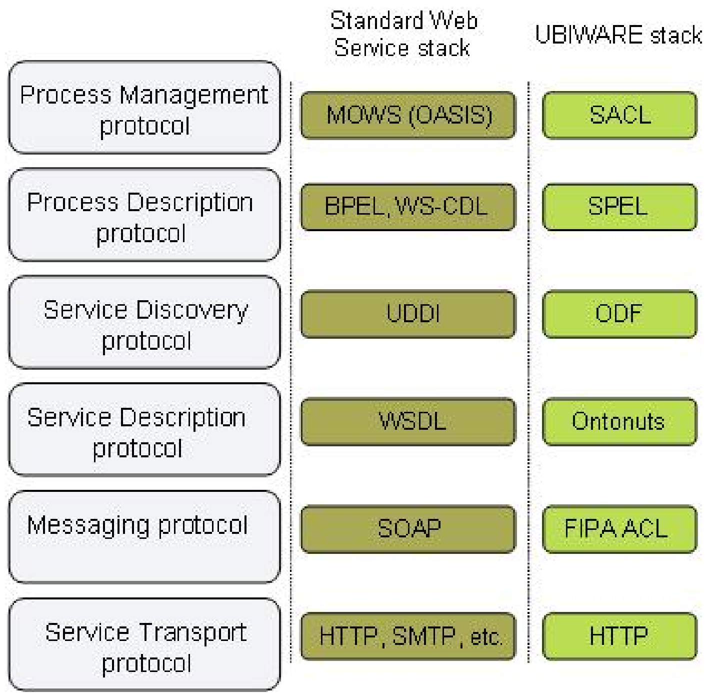
Standard Web Service stack
UBIWARE stack
Process Management protocol
MOWS (OASIS)
SACL
Process Description protocol
BPEL, WS-CDL
SPEL
Service Discovery protocol
UDDI
ODF
Service Description protocol
WSDL
Ontonuts
Messaging protocol
SOAP
FIPA ACL
Service Transport protocol
HTTP, SMTP, etc.
HTTP

***Figure 1.1** – Web Services stack vs. UBIWARE.*

In UBIWARE the bottom two layers of this picture (Service Transport and Messaging) are provided by Jade agent platform which is a basement of UBIWARE platform. The modeling of service description and service discovery layers is presented in the following subsections. The Process management and configuration layer is described in WP4. The abbreviations SPEL (Semantic Process Execution Language) and SACL (Semantic Agent Configuration Language) constitute parts of the framework that were not included in the original research plan, but will be elaborated in the subsequent development activities. Although, the S-APL platform language already includes process execution constructs and mechanisms as well as configuration capabilities using meta-rules, still, we have recognized the need for these extensions with the growth of the platform capabilities and collection of user experiences. New formalisms will target the developers' group and will simplify the definition of processes and configurations within the platform by introducing intuitively understandable models for these.

[1] http://roadmap.cbdiforum.com/reports/protocols/

1.2 Externalizing Ontonuts

The Ontonut instance describes a capability with known required input and expected output. In order to make it published as a service to the external world, the description should be extended to enable platform wide Ontonuts advertisement and discovery. We introduce new entities and properties to the Ontonuts ontology for component publishing and control. We also add comprehensive information for access and invocation. The Ontonut class has been extended with the wrapper subclass *ServiceNut*, enriched with the following properties:

- Access mode (public or authorized)
- Publishing mode (Direct marketing, Big board or Billboard)
- Access point (unique agent name and Ontonut name within the agent)
- Contracting (QoS, SLAs, pricing, payment methods, etc.)

These entities may look similar with their WSDL analogs up to the input and output level where the main difference shows up. We benefit from features of S-APL language, which being actionable, introduces also mechanisms for semantic reasoning and semantic matchmaking.

1.3 Advertising Ontonuts

As soon as a component is supplied with the proper annotation, the component owning agent can inform other agents about appearance of a new service. We distinguish three possible ways of advertisement within the platform (see Figure 1.2):

- Direct marketing (tell to chosen agents)
- Big board (publish to directory facilitator)
- Billboard (publish to the environment)

In the direct marketing approach each agent possesses the functionality of Directory Facilitator. Agents can advertise their services to each other and serve distributed service search requests. Such scheme fully follows peer-to-peer philosophy.

In the Big board scheme agents use Ontonuts Directory Facilitator (ODF) agent as a registry of agent services. The ODF represents centralized approach and serves similar purposes to UDDI registry in WS-stack. The ODF is implemented as an agent role; therefore the agents should request from the platform Directory Facilitator the ODF agent name. As soon as the name is resolved, the agents can start advertising their services and search for services advertised from others. ODF agent acts in parallel with the Directory Facilitator Jade agent. Whereas DF takes care of agent roles, the ODF takes care of UBIWARE agent services. The platform may have several ODF agents. In such case, the agents should have additional ODF selection criteria in order to find the proper ODF. On top of basic functionality such as registering and deregistering agent services, ODF may provide subscription function. This function is useful for time-critical processes. The subscription guarantees that the subscriber agent will be informed if a service provider agent has terminated unexpectedly without prior notice and/or deregistration from ODF.

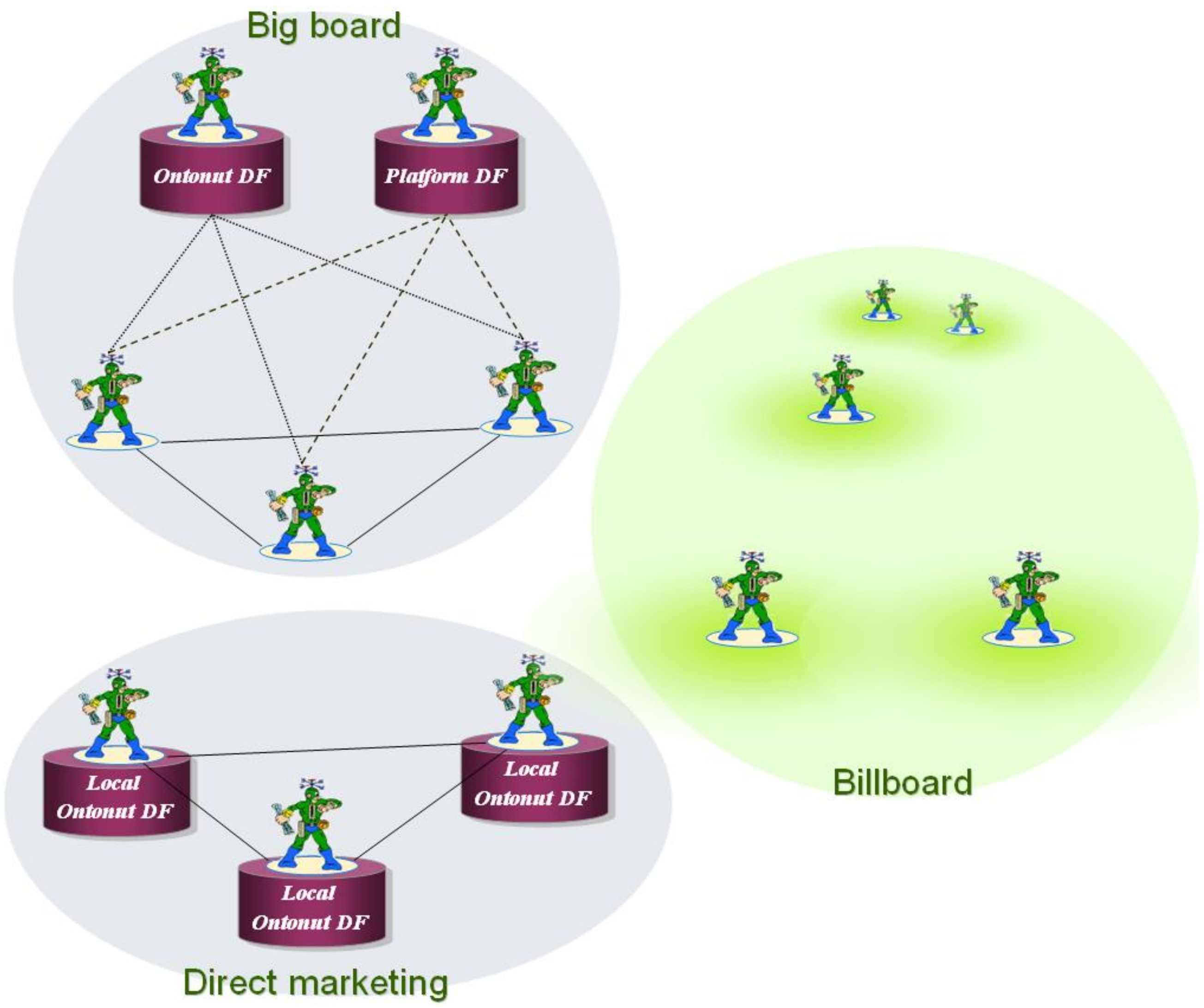

Figure 1.2 – *Ontonut advertising schemes.*

The Billboard scheme is based on the conceptual abstraction of Agent Observable Environment (AOE). The UBIWARE AOE has been introduced in the 2nd year deliverable (D2.1, 2008). Within such conceptual abstraction the agents can determine the proximity to each other to the environment criteria. The agents can sense service advertisement in a certain area around the advertising agent.

Nevertheless, each scheme described above requires a certain service description to be published.

Publishing Ontonuts

The publishing process may be done in two ways:

- By default (assign a property value of a respective Ontonut as: "*O1 di:published di:true*")
- Explicit (Define wrappers and their parameters explicitly)

The default publishing heavily relies on the Ontonuts engine configuration. Default wrappers will be generated automatically by the Ontonuts engine in accordance with the engine settings. The engine, for example, by default may define public access with the high QoS which may result in high computational load, therefore this mode should be treated carefully.

The access mode of the service may require authorization or registration before the actual service use. The prerequisite for service use may, for example, be set to "*Client agent should belong to the corporate group*".

Depending on the publishing mode, the service description is sent either to ODF, or to peer agent, or put as visible to the environment.

The unique identifier of the published service should stay the same during the whole service life cycle. We compose the service uid from the organization URI, where service is published or provided, plus the unique name of the agent role within the organization, plus the service version and local service name. Such composition should ensure the unique combination, at the same time allowing the agent to be disabled and started up again with the same service UID. For example, pdf-converting service provided by Industrial Ontologies Group, is run by the agent locally called "publisher". The service UID in such case would be

"*www.iog.jyu.fi/publisher/ver1.0/PDFConvert*".

Contracting is an important part of any business environment. The business scenarios within UBIWARE will require mechanisms for presenting the comprehensive contracting details. At this stage, we include most notable concepts introduced by IBM in their WSLA language[2]. Service Level Agreements are also a subject of research in FP7 program, e.g. SLA@SOI project[3]. The servicing model may later require further extension; however, we think that detailed contracting mechanism specification should be deeply elaborated if UBIWARE platform will enter the commercialization phase. At present, we include following items into contract specification (Figure 1.3).

The service provider and consumer should be clearly identified as well as the contract version and the version of the service to be provided. We also include a contact entity of the provider, i.e. an agent to be contacted by the consumer in any circumstances. The definitions and obligations include the interfaces (operations in service description) as well as test cases to them. We omit protocol definitions as far as we assume that agents offer services within the platform, hence the communication medium is available. The security constraints may require password protected authentication or group membership.

The service description (*ServiceNut*) includes also the functional description as a reference to the Ontonut with definition of input and output. We also take into account those cases, where service may offer a set of capabilities; therefore, service may link to several Ontonuts.

[2] http://www.research.ibm.com/wsla/
[3] http://sla-at-soi.eu/

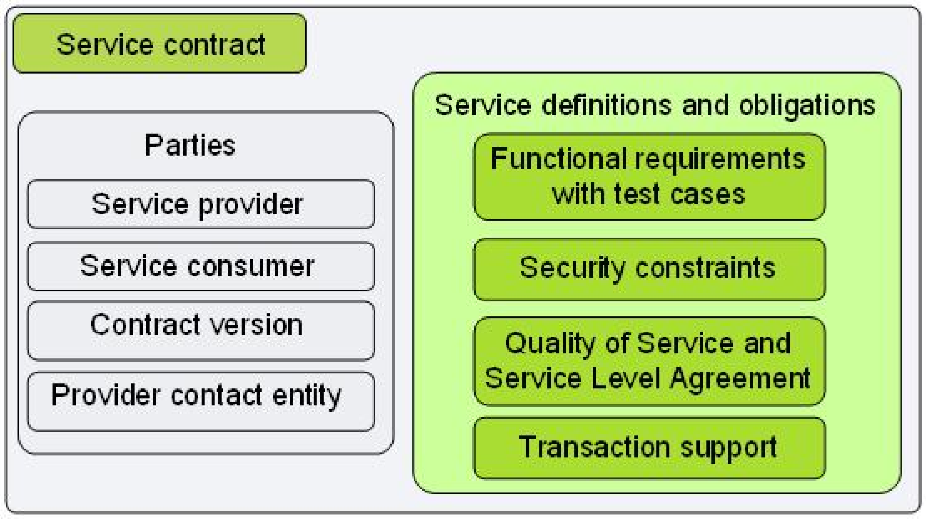

Figure 1.3 *– Service contract specification.*

The service publishing process can be organized in two modes. First mode is a direct capability publishing. The publishing agent sends the request to the ODF of the following form:

```
I want {
   You publish {I haveService {
      accessMode is public
      serviceURI is www.iog.jyu.fi/service/id1 //Access point
      input is {semantic pattern}
      output is {semantic pattern}
      …
   }}
}
```

The Ontonuts Directory Facilitator agent treats such request for publishing as an access point and semantic description in one place. The semantic search and matchmaking of services in this mode is performed on the fly by the ODF.

In the second mode, the publishing message is extended with the reference to the abstract service description. The abstract service descriptions can be introduced to improve the search efficiency in domains, where number of capabilities offered by agents is big. Therefore the access points (agent services) are bound to abstract service definitions. These definitions can be organized into the domain service ontology and later be used in the planning of abstract processes. The abstract service definition specifies input and output patterns and may omit other properties. When publishing service which is referring to the abstract service definition, the publishing message will not have big syntactical difference, as it will just acquire additional property:

```
  abstractServiceURI is ?auri
```

Although the abstract service includes input and output definitions, we still keep input and output patterns in the service description, because these descriptions may vary in detail level, however, they should obey class-subclass relationship. Such definition may seem redundant, but we think that it may improve planning and avoid overheads in process execution. We discuss it further in this paper.

Discovery and subscription

Any agent can send search request to the ODF. Typical request may have following form:

```
I want You answer {
?agent hasService {
        input is {pattern}
        output is {pattern}
        serviceURI is ?uri
        abstractServiceURI is ?auri
        serviceCategory is ?category
            …
         }
}
```

As a search criterion, the requestor may use a URI of the abstract service description, or a category name. The category reference is a property of abstract service description and may be used in search cases, when input and output can not be properly specified. In such case, the search function is somewhat similar to the standard service discovery in UDDI. Categorization of services is a domain specific task therefore we leave it out of scope here. As a search result the ODF agent returns Ontonut services that match the search request. In terms of SOA, the ODF returns a list of endpoint entries (with service bindings). In case, when a search criterion defined as input and output patterns, the search procedure performs matchmaking. The search patterns provided are semantically matched against available abstract service descriptions and the descriptions, in turn, have service implementations (access points) bound to them. The semantic descriptions of services both abstract and real ones fall into the problem of pattern matching and completeness of the description.

Well-known approaches in Semantic Web Service discovery like WSMO[4] suggest that abstract semantic service descriptions should be defined with low detail level to keep semantic reasoning and matching fast enough. The real world services are then bound to the abstract descriptions and the service consumption itself may be needed to clarify if the service fits the goal.

Another option is to define in the design phase the annotations of services (bound service capabilities) as precisely as possible (all constants known before execution must be included into the description). Abstract capabilities in turn, should be defined in a way that makes it suitable for a process designer (human) to build business process logic. The attainability of the abstract process then can be checked against available bound capability descriptions. The reason for such detailed annotation is to avoid frequent null results in the runtime. In the planning phase, detailed description is used to detect the candidate process chains that will always result in null (empty set) because the dependable variables' values may never match within the whole process chain, e.g. the sequence

```
(1) A A ?a => B B ?a,
(2) B B ?a => C C ?a
```

[4] http://www.wsmo.org/

may never have successful result (C C ?a), when first capability gives e.g. ?a = 1,2,3, whereas second capability works only with the value range from 5 to 10. Therefore, precise definitions make planning more efficient and increase the probability of non-null result (some null-resulting plans can already be excluded in the planning stage).

It is also possible to supply capability description designer with the automated tool that would allow extraction of static data values from the underlying sources or code.

In case of service matching, if a set of matching service providers was found, an agent can choose best suitable alternative by negotiating with the service provider agent and then request ODF for subscription for the service (and the provider) chosen. The mechanism of service discovery, however, heavily depends on the approach to service annotations and may fall into two stages of service discovery and process planning:

1. Find potentially suitable services for the process (semantic matching of abstract service ontology artifacts and the goal specified)
2. Negotiate with the potential service provider candidates on the details of service execution (pre-execution). For example, ask if the service provides flight bookings from Moscow to Paris at all, or if the payment can be performed with Visa card.

Therefore, if the abstract capabilities will cover generic service classes, then the search and planning may take these two stages.

Nevertheless, other scenario options are possible. A service provider may submit detailed service description if an ODF supports it. In this case, the second stage mentioned above can be done without direct service contact. For example, if a database provides additional metadata information on the value range of the parameters in it or some statistical calculations, like mean value of certain parameter, then the planner can take into account value ranges without preliminary database querying.

We can distinguish between consuming informational part of the service (negotiation, when the physical state of the world does not change) and consuming the irrevocable service part that changes the state. Good example of both can be: collecting information about the flights available and booking the flight. The informational part of the service can in some cases be presented as an extended semantic annotation and may reside outside of the service (for example, the service registry may contain up-to-date information about destinations of airline company, whereas available seats may only be requested from the service itself). Therefore, the consumption of the service may not really happen unless the irrevocable part or time-critical informational part is used (e.g. ticket booking and seats available).

1.4 Conclusions and further work

Service advertising and discovery are an intrinsic part of the process management scenarios. In the open environment the value of it should not be underestimated. With the adoption of automated contracting and legal mechanisms in the services world, the dynamic matchmaking and service discovery will become a cornerstone of business process management. We perceive that process management will be driven by autonomous entities with certain degree of freedom to take decisive actions. The foresight of the future leads us towards further directions the dynamic process management area.

In particular, within the framework defined in the introductory part of this Section we will define behavioral patterns and data structures that specify how the agent should (re-)act in order to support servicing agent interaction scenarios. The scenarios are domain independent and constitute three infrastructural layers: execution layer, process management layer and configuration layer.

The execution layer defines how one agent can request another agent to execute an action and receive action results.

Process management layer is built on top of the execution layer and allows agents to control composite actions in runtime, e.g. before the actual execution starts, the process handler agent can request for availability of candidate agents, if they confirm participation in the process or not. This layer also involves contracting.

The configuration layer specifies a meta-level for agent-to-agent interactions that allows agents to negotiate about the processes they run and cooperatively agree on changes.

UBIWARE Deliverable D3.1:
Workpackage WP2:
Task T3.1_w2:
Workpackage leader: *Sergiy Nikitin*

2 UbiBlog – Managing Distributed Resource Histories

During WP2's Year 2 (the *Integration* phase), we have realized the possibility of querying a set of distributed, autonomous and semantically heterogeneous resource histories as they were one virtual database. We have introduced a new concept of *Ontonuts* and a corresponding technology. The Ontonuts where tailored to solve the problem of distributed querying in the first place. However, we have generalized the concept to the level of semantic capabilities, which allows us to use goal-based dynamic planning and execution of agent's actions. Therefore, the third year phase of the WP2 (Mining phase) as it was specified before would rather be an application case of the Ontonuts technology than a research topic. With respect to this fact we have decided to concentrate our efforts towards the fusion of the WP1 and WP2 3rd year objectives. Shortly, the WP1 objective for the 3rd year is to elaborate a mechanism for advertising of agent capabilities and role scripts (complex capabilities) amongst agents, whereas WP2 research targets data mining capabilities only. Therefore, it is reasonable to integrate WP1 and WP2 under one umbrella of Ontonuts technology which is targeting even more ambitious goal – to enable automated (re)planning and execution of semantically annotated agent actions including distributed data querying, data mining as well as distributed agent-to-agent servicing.

2.1 Servicing Data Mining in UBIWARE

This section demonstrates the data mining capabilities as services. The project plan for the 3rd year combines the WP1 and WP2 where WP1 should provide a technology for the WP2 to be developed as a case study. Therefore, we use service orientation for both data source as well as data mining capabilities. To model the data mining services we have to define a corresponding data mining domain ontology. The ontology will cover data mining methods and requirements for method inputs and respective outputs. The inputs and outputs should, in turn, refer to the data types. The data mining domain should be fine-grained; hence it can not include all possible applications of its methods. Also, we should keep the granularity of the conceptualization. Therefore, we have to distinguish between the data mining models and their application scenarios.

2.2 Standardization efforts in data mining models

The efforts towards standardization of data mining techniques, methods and formats have been a matter of discussion for the last ten years. Java community introduces JSR 73 specification called Java Data Mining API, which defines mining models, data preparations and data mining results. Data mining software market is very diverse ranging from commercial corporate licenses (SAS[5]) up to freely available tools, e.g. Weka[6] (Witten and Frank, 2005). From the UBIWARE perspective we take special attention to the standardization efforts in the data mining domain as one of the important categories in business intelligence. One of the notable actively developed standards is PMML language[7], (Guazzelli et al., 2009). The language is a standard for XML documents which express instances of analytical models. In our work we take PMML as a reference model for the Data Mining Ontology and enhance both the model as well as the data with the semantic descriptions required to automate data mining methods application to the domain data. We do not take into account the stage of information collection, preparation, etc. We assume that data is ready for data mining algorithm application. The PMML structure for model definition is composed of a set of elements that describe input, model and outputs (Figure 2.1).

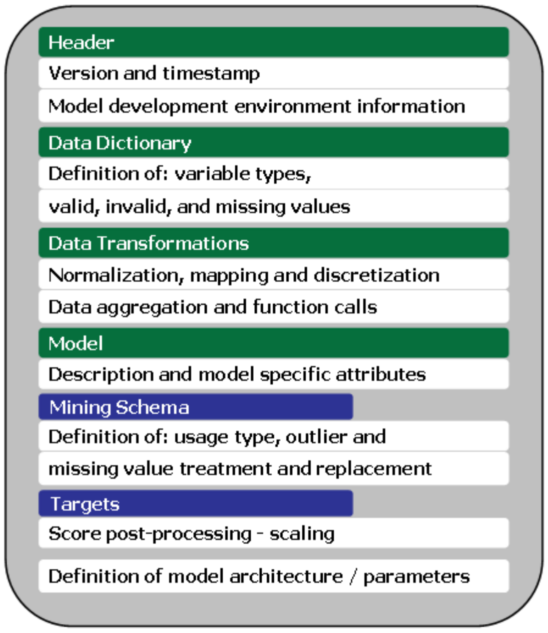
Header
Version and timestamp
Model development environment information
Data Dictionary
Definition of: variable types,
valid, invalid, and missing values
Data Transformations
Normalization, mapping and discretization
Data aggregation and function calls
Model
Description and model specific attributes
Mining Schema
Definition of: usage type, outlier and
missing value treatment and replacement
Targets
Score post-processing - scaling
Definition of model architecture / parameters

***Figure 2.1** – PMML structure*[8].

[5] http://www.sas.com/
[6] http://www.cs.waikato.ac.nz/~ml/weka/
[7] Data Mining Group. PMML version 4.0. URL http://www.dmg.org/pmml-v4-0.html
[8] http://knol.google.com/k/-/-/3pz0mz6zvkz16/74ajfe /rpmmlfig1.png

The schema for the root element of the PMML document is defined as follows:

```
<xs:element name="PMML">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Header"/>
        <xs:element ref="MiningBuildTask" minOccurs="0"/>
        <xs:element ref="DataDictionary"/>
        <xs:element ref="TransformationDictionary" minOccurs="0"/>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
         <xs:choice>
            <xs:element ref="AssociationModel"/>
            <xs:element ref="ClusteringModel"/>
            <xs:element ref="GeneralRegressionModel"/>
            <xs:element ref="MiningModel"/>
            <xs:element ref="NaiveBayesModel"/>
            <xs:element ref="NeuralNetwork"/>
            <xs:element ref="RegressionModel"/>
            <xs:element ref="RuleSetModel"/>
            <xs:element ref="SequenceModel"/>
            <xs:element ref="SupportVectorMachineModel"/>
            <xs:element ref="TextModel"/>
            <xs:element ref="TimeSeriesModel"/>
            <xs:element ref="TreeModel"/>
          </xs:choice>
        </xs:sequence>
        <xs:element ref="Extension" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
```

The models defined in the PMML specification ver. 4.0 are listed as choice options, whereas one document may include more than one model. The specification provides means for exhaustive model description, thus the model can be fully exported or imported without losses. Such model transportability gives huge opportunities for service orientation of the data mining methods. We can also expect the PMML models reuse in the cloud computing domain in the nearest future. Cloud Computing defines a stack of abstract layers: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). A combination of all the layers defines a configuration of the cloud. The specification of a software-independent descriptive data mining standard implies that Infrastructure and Platform layers are fully transparent for the data mining services, i.e. the functional characteristics of the services will be same for any stack configuration. The QoS, however, may vary depending on the performance of the hardware and efficiency of platform software.

2.3 Classification of data mining services

The upper ontology of data mining services is shown on Figure 2.2

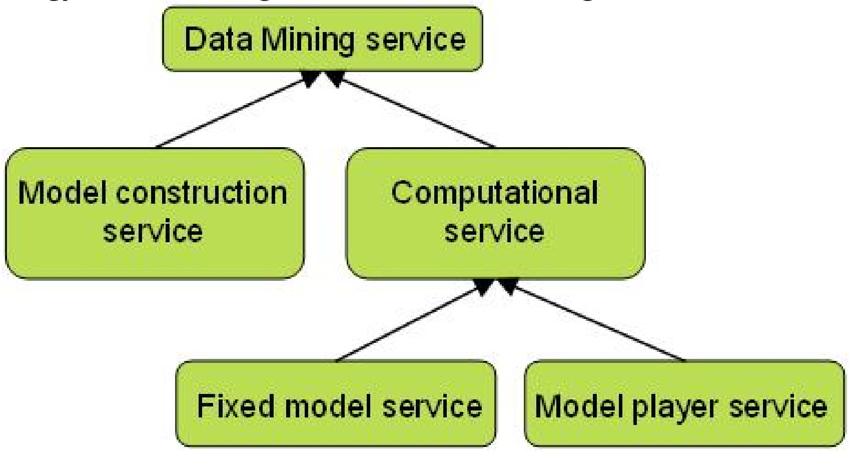

***Figure 2.2** – Upper ontology for data mining services.*

We consider two major categories of data mining services:

- model construction services
- computational services

The model construction services produce a model (a semantic description) from the set of learning samples. In other words, input of such a service is learning data and conditions (for the neural network depending on its mode it can be a set of vectors plus e.g. initial network parameters). The output of the model construction service is a model with the parameters assigned (e.g. a neural network model, see Figure 2.3).

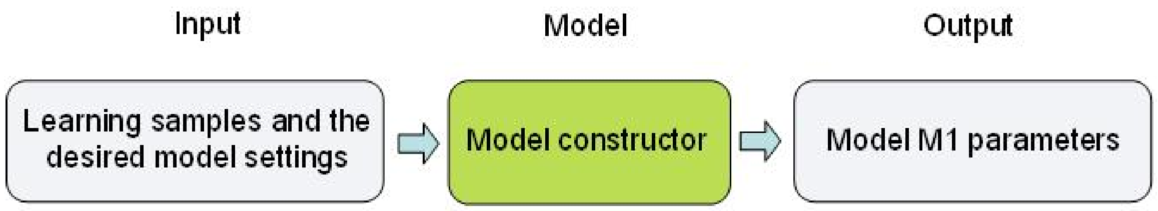

***Figure 2.3** – Model construction service.*

The group of computational services can also be divided into two major groups:

- services with fixed model
- model player services

The services with fixed logic define the format of the input and output as well as provide reference model description and the parameters that determine how the model is configured. For example, Figure 2.4 shows the definition of the neural network-based alarm classifier service for a paper machine.

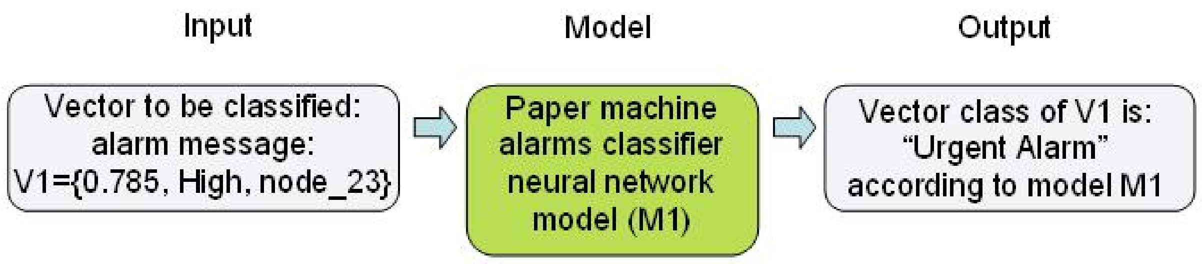
Input
Model
Output
Vector to be classified: alarm message: V1={0.785, High, node_23}
Paper machine alarms classifier neural network model (M1)
Vector class of V1 is: "Urgent Alarm" according to model M1

***Figure 2.4** – Neural network model in classification service.*

Usage of model player services has two stages: in the first stage the service accepts as an input the service model and then in the second stage it can serve as a fixed model service (see Figure 2.5).

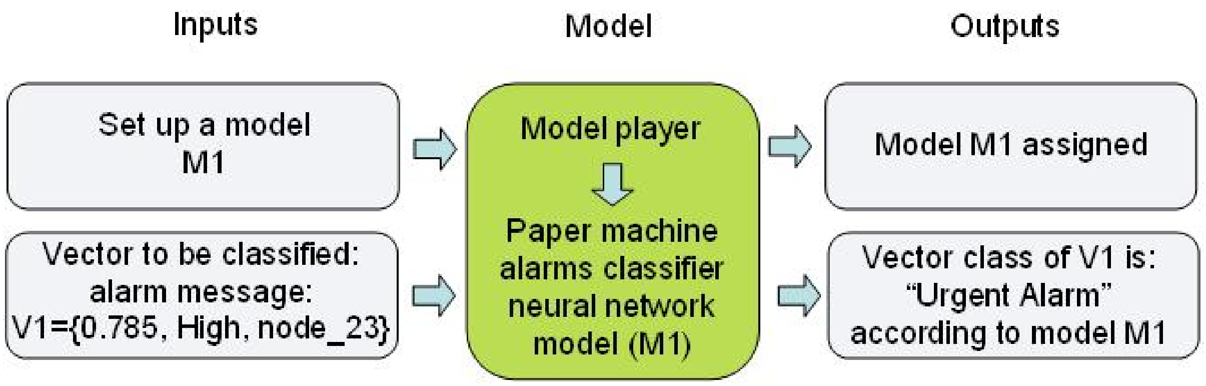
Inputs
Model
Outputs
Set up a model M1
Vector to be classified: alarm message: V1={0.785, High, node_23}
Model player
Paper machine alarms classifier neural network model (M1)
Model M1 assigned
Vector class of V1 is: "Urgent Alarm" according to model M1

***Figure 2.5** – Model player service.*

The true power of data mining services can be demonstrated in combination with the distributed querying, data mining model construction and further classification. The generic use case of such combination is shown on Figure 2.6.

The automated data collection process (first step in the use case) has already been researched in the (Bleier et al., 2008). The approach we have developed allows dynamic distributed query planning and execution. As soon as data is collected (usually in form of a table of query results), it may undergo the preparatory step to become learning data set. We omit the procedure of normalization here, or other data transformations, however, they will be necessary and obligatory. As soon as the learning set is ready, a desired model constructor should be chosen (step 2). The model constructor may require specific data preparation, therefore it is good to combine data preparation step with the model constructor service. As an input, model constructor may require additional input parameters for model building. Those may be set as default, or, if other parameter were prepared, they should be supplied in the proper form. When a model is ready, we feed it to the model player service which is a platform service in terms of cloud computing, because it provides an infrastructure and software platform for service execution. As soon as our newly built model is deployed as a service, we can start classifying the data vectors, e.g. alarms coming from paper machine.

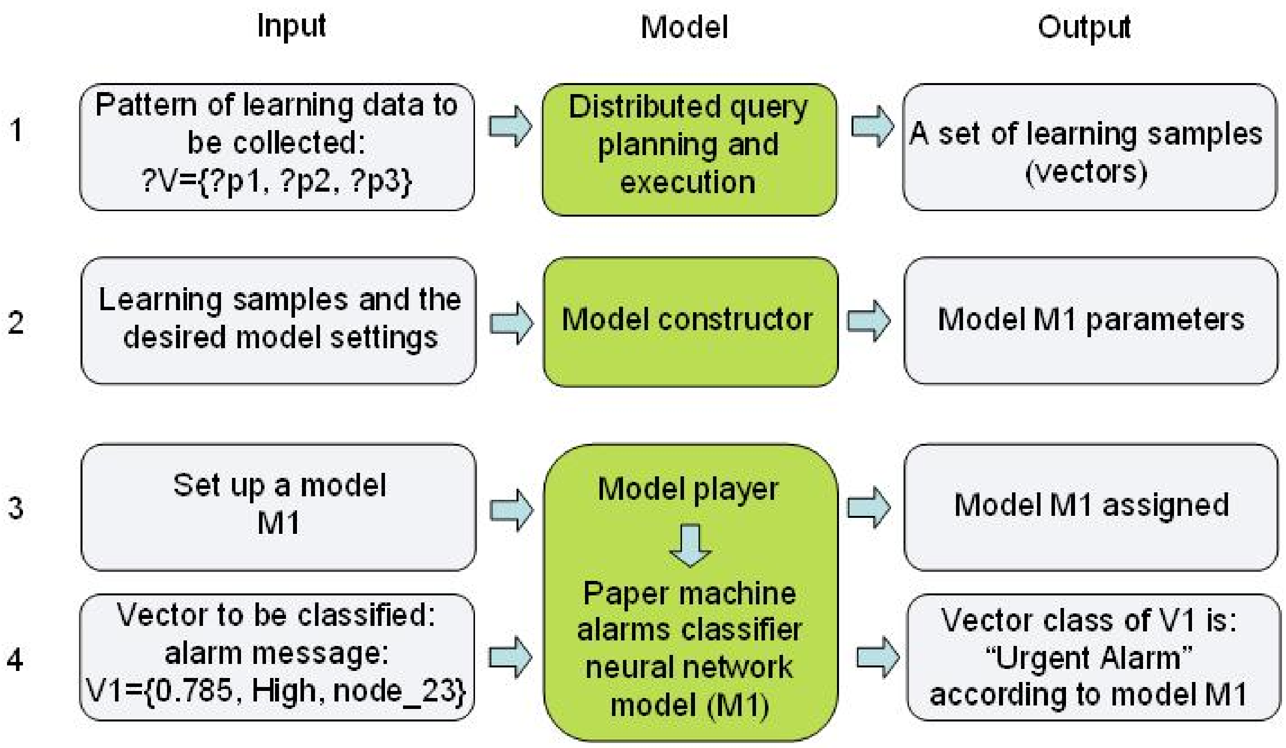
Input
Model
Output
1
Pattern of learning data to be collected: ?V={?p1, ?p2, ?p3}
Distributed query planning and execution
A set of learning samples (vectors)
2
Learning samples and the desired model settings
Model constructor
Model M1 parameters
3
Set up a model M1
Model player
Model M1 assigned
4
Vector to be classified: alarm message: V1={0.785, High, node_23}
Paper machine alarms classifier neural network model (M1)
Vector class of V1 is: "Urgent Alarm" according to model M1

***Figure 2.6** – Querying-Learning-Classification use case.*

2.4 Conclusions and future work

The research presented above describes specific domain of data mining services. We foresee that model player services will be a successful business case for the emerging paradigm of cloud computing. Pay-per-use principles combined with high computational capacities of cloud and standardized DM-models will be definitely an alternative to expensive business intelligence and statistics toolkits.

Another niche of data mining services in cloud computing can be model construction services. Such systems will drive innovations in data mining methods as well as applied data mining in certain domains. Such service will compete by introducing know-how and innovative tools and algorithms that bring add-values in e.g. predictive diagnostics or computational error estimation. This direction will lead to so-called "web of intelligence"[9].

The role of UBIWARE in cloud computing emerges as a cross-cutting management and configuration glue for interoperability of future intelligent cloud services (see Figure 2.7).

The main burden of UBIWARE will be management of consistency across different domain conceptualizations (Ontologies) and cross-domain middleware components. Fine-grained ontology modeling is still a challenge for research community and we predict that in the nearest future the domain modeling will be task-driven, i.e. the domain model engineers may

[9] Terziyan, 2009, www.cs.jyu.fi/ai/ICNS-2009.ppt

incorporate some standardized and accepted conceptualizations, whereas the whole ontology for solution will be tailor made. Tailored ontologies will require subsequent mapping mechanisms and additional efforts. Nevertheless, we have to cope with it because building one centralized world ontology has been reasonably criticized as utopia.

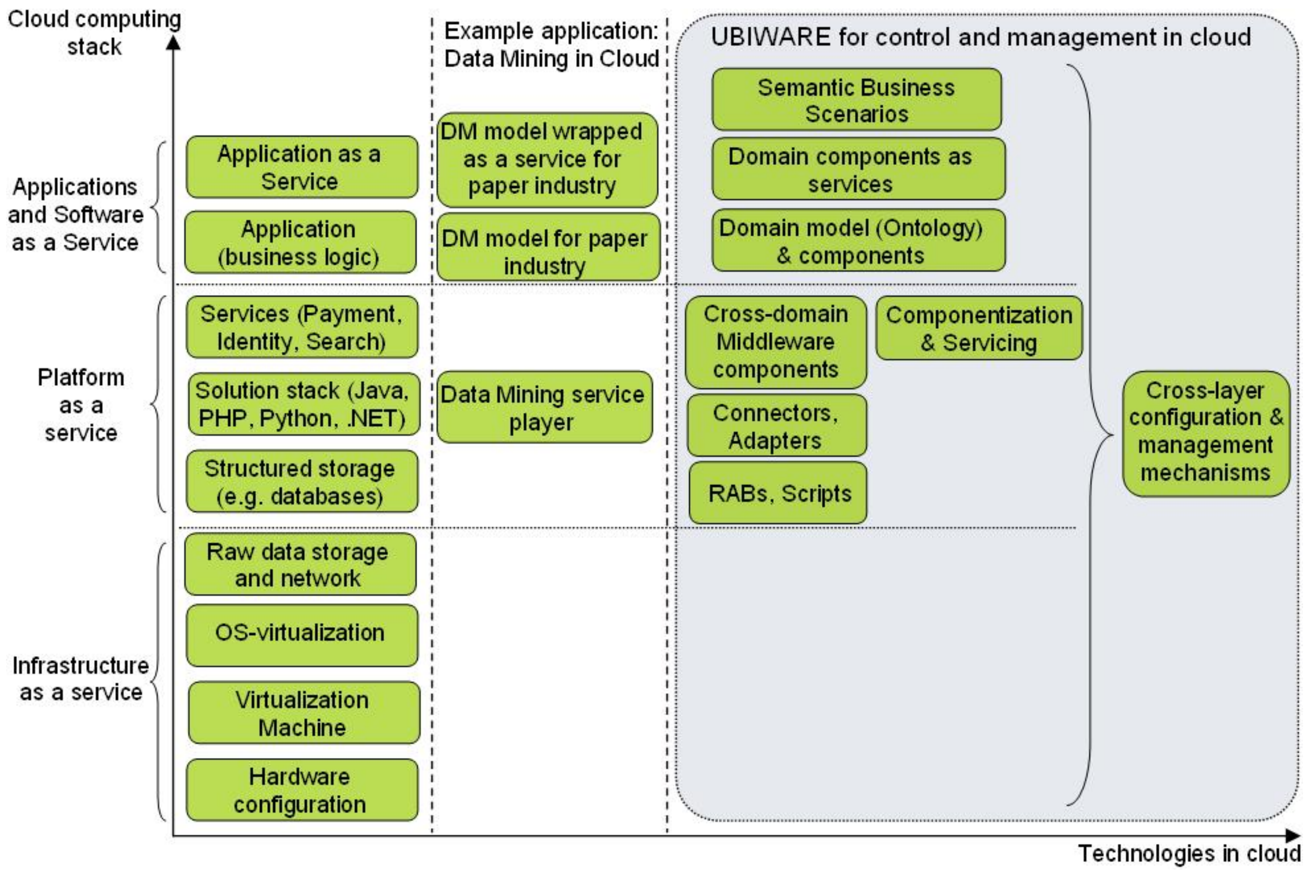
Cloud computing stack
Applications and Software as a Service
Platform as a service
Infrastructure as a service
Application as a Service
Application (business logic)
Services (Payment, Identity, Search)
Solution stack (Java, PHP, Python, .NET)
Structured storage (e.g. databases)
Raw data storage and network
OS-virtualization
Virtualization Machine
Hardware configuration
Example application: Data Mining in Cloud
DM model wrapped as a service for paper industry
DM model for paper industry
Data Mining service player
UBIWARE for control and management in cloud
Semantic Business Scenarios
Domain components as services
Domain model (Ontology) & components
Cross-domain Middleware components
Componentization & Servicing
Connectors, Adapters
RABs, Scripts
Cross-layer configuration & management mechanisms
Technologies in cloud

Figure 2.7 *– Role of UBIWARE in cloud computing.*

UBIWARE Deliverable D3.1:
Workpackage WP3:

3 SURPAS – Smart Ubiquitous Resource Privacy and Security (*postponed*)

The security is often seen as an add-on feature of a system. However, in many systems (and UBIWARE is one of them), the system remains nothing more but a research prototype, without a real potential of practical use, until an adequate security infrastructure is embedded into it. The main objective of this work package is the design of the SURPAS infrastructure for policy-based optimal collecting, composing, configuring and provisioning of security measures in multi-agent systems like UBIWARE. SURPAS follows the general UBIWARE vision – configuring and adding new functionality to the underlying industrial environment on-the-fly by changing high level declarative descriptions. Regarding security, this means that SURPAS will be able of smoothly including new, and reconfiguring existing, security mechanisms, for the optimal and secure state of the UBIWARE-based system, in response to the dynamically changing environment. The optimal state is always a tradeoff between security and other qualities like performance, functionality, usability, applicability and other.

Questions, related to this workpackage, are however already partially answered during previous project years, and will also be treated on the more general level of organizational policies of any kind in WP1. Because of that and because of reduced funding, it is considered to be reasonable not to perform work in this WP during Year 3.

UBIWARE Deliverable D3.1:
Workpackage WP4:
Task T3.1_w4:
Workpackage leader: *Michal Nagy*

4 Self-Management, Configurability and Integration

Self-management and configurability in UBIWARE can be seen from two points of view:

- Initial self-configuration
- Runtime self-configuration

Initial self-configuration is understood as the ability of the system to interconnect and configure its components based on a certain goal or policy specified by the user. After specifying the goal of the system, the system itself should be able to automatically choose proper agents and delegate proper roles to them. The result should be a system that is performing the task specified by the goal. The user does not have to provide the system with any code, only the goal is needed. The system will find the best configuration based on the goal specified and a domain specific ontology.

Runtime self-configuration is the ability of the system to adapt to the environment. Thanks to this ability the system is able to perform its task even if the circumstances change. The process of runtime self-configuration should be context-aware, ontology-driven and policy-based.

Based on the text above, we specify the following research questions. First two are related to initial self-configuration and the third question is related to runtime self-configuration.

- How can we find suitable agents to perform the task based on the goal specified by the user?
- How can we transform a goal into a set of scripts to be executed by these agents?
- Once the system is configured, how can we maintain this state even if the circumstances of the system change?

4.1 Introduction

The vision of autonomic computing describes self-management as the ability to maintain and adjust operation of system components (Kephart and Chess, 2003). It distinguishes four main aspects of self-management – self-configuration, self-optimization, self-healing and self-protection. Self-configuration deals with automated configuration of components and systems following high-level policies. The goal of self-optimization is continual seek of improvement of performance and efficiency. The purpose of self-healing it to detect, diagnose, and repair localized software and hardware problems. Self-protection gives the system the ability to automatically defend against malicious attacks or cascading failures.

According to Kramer and Magee (Kramer and Magee, 2007), a self-managed architecture is one in which components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goals of the system. In their work they don't explicitly mention the ability of the system to protect itself against malicious attacks. We also believe that in the initial phase security issues should be left alone.

In this workpackage we will look at the issue of initial configuration and runtime self-configuration from the process perspective. This perspective corresponds to work done in workpackage 1. The issue of initial configuration involves configuration of an abstract process. In the second issue we use the term “self-configuration”, however our system also involves a portion of self-healing and it may involve self-optimization as well. Therefore we will also use the term self-management.

In the original research plan we call these capabilities initial self-configuration and runtime self-configuration. In broader sense initial configuration can be understood as self-configuration as well, however this type of configuration is a one-time action. Self-configuration is considered a continuous process and therefore we decided to rename it to initial configuration.

In our work initial configuration is understood as the ability of the system to interconnect and configure its components based on a certain goal or policy specified by the user. After specifying the goal of the system, the system itself should be able to automatically choose proper agents and delegate proper roles to them. This delegation and redistribution is done under certain circumstances defined by the context. We understand the context as any information that can be used to characterize the situation of an entity (Dey, 2001). An entity is the process, user or any object that is considered to be relevant to the interaction between the user and the process.

Runtime self-configuration takes its place after the initial configuration took place and execution of the process started. The initial configuration was performed with respect to a certain context valid at that time. In general, a process is a long-running action. Since the context is dynamic and the process may be long-running, the context may change during the process execution. If the context change is significant, it may affect the way the process should be configured. Also, during the process execution a component may fail. Runtime self-configuration is trying to handle exactly these situations.

4.2 Assumptions and requirements

We assume that every process has exactly one agent acting as a process director. Its goal is to perform initial configuration of the process, communicate with the user (if necessary) and after a successful configuration, start the process execution.

Later we assume that there is a pool of different abstract processes. This pool is accessible by the director agent. An abstract process consists of starting point, ending point and several abstract steps interconnected between each other. The starting point cannot have any inputs and the ending point cannot have any outputs. Every abstract step is semantically described using. This description contains information about what should be done in this step, however it doesn't say how it should be done. This is the reason why the step and the whole process are called abstract.

In the system there is a repository storing information about available components implementing a certain functionality. In Ubiware these components are called Ontonuts. According to work package 1 terminology, this repository is called ODF (Ontonuts Directory Facilitator). ODF stores a set of Ontonut descriptions including the address of an agent that is handling this Ontonut. In addition to functional description of an Ontonut (input, output, etc.), there can be a non-functional description included (e.g. time, price, etc.). The environment with all mentioned components can be seen in Figure 4.1.

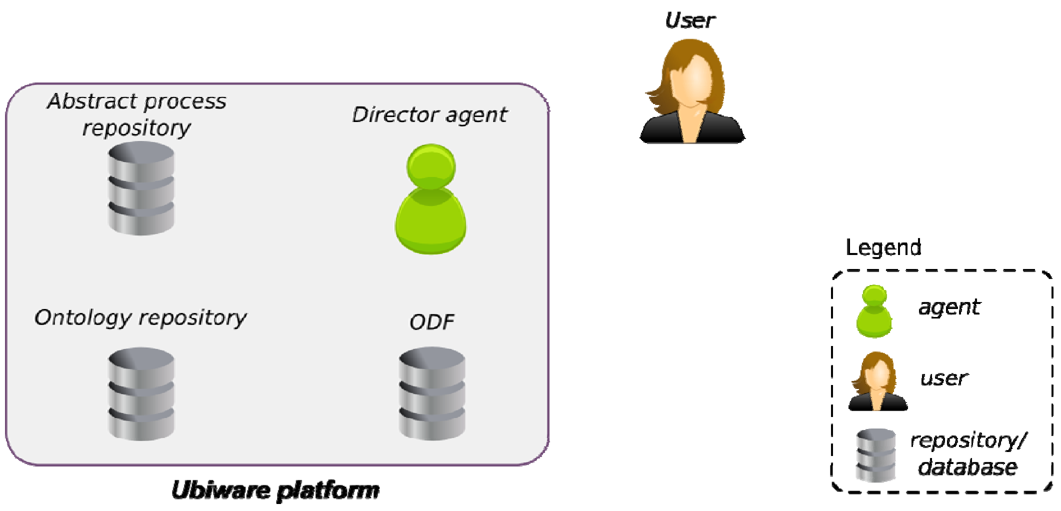

Figure 4.1 *- Ubiware environment for self-configuration.*

We also have some requirements on components. We expect that in case of an error every component will end its execution gracefully and report an error. However, this might be difficult to enforce. We discuss this issue in the Section 4.5.

4.3 Initial configuration

4.3.1 Abstract process

As mentioned earlier, the purpose of the initial self-configuration is to provide a configuration description for an abstract process and thus create an executable process. An executable process already consists of instantiated steps. That means it contains all the necessary bindings to real Ontonuts that will implement a particular step within the process. The relationship between an abstract process and an executable process is depicted in Figure 4.2.

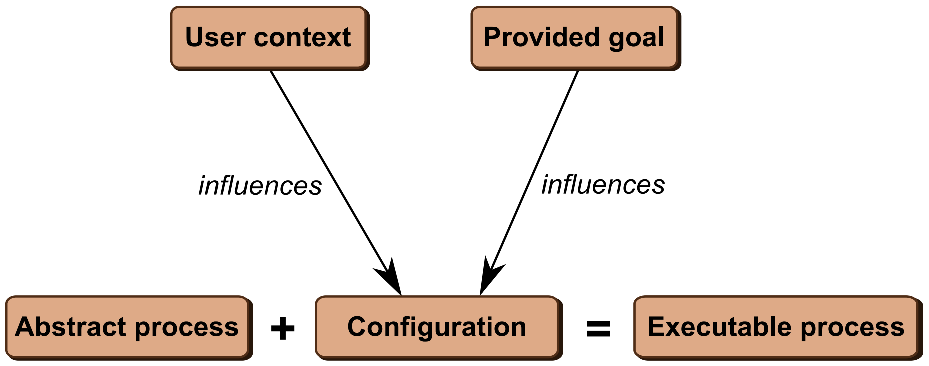

***Figure 4.2** - The relationship between an abstract process and an executable process.*

The abstract process is semantically described. Every process has a start and an end (or possibly more ends) and a set of abstract steps that reach the end. At the end a goal is semantically specified (which the process leads to). An example could be an abstract process of ordering flowers for somebody (Figure 4.3). This abstract process could carry a human-readable description “Delivery of some flowers to some person”. It doesn't put any restrictions on flower type or other attributes.

This abstract process has an input where the following process variables have to be defined:

1. Person first name
2. Person surname
3. Flower type

The goal of the process is to create a relationship of type `c:ThingSent` between an object of type `c:Flower` and an object of type `c:Human`. The abstract process consists of two abstract steps. In the first step the address `e2` of a person `e1` is retrieved. As mentioned earlier, this abstract step doesn't say anything about how this address should be obtained. The second abstract step defines that the flower should be sent to the address from the previous step. After this step is executed, the goal is fulfilled and the flower is delivered.

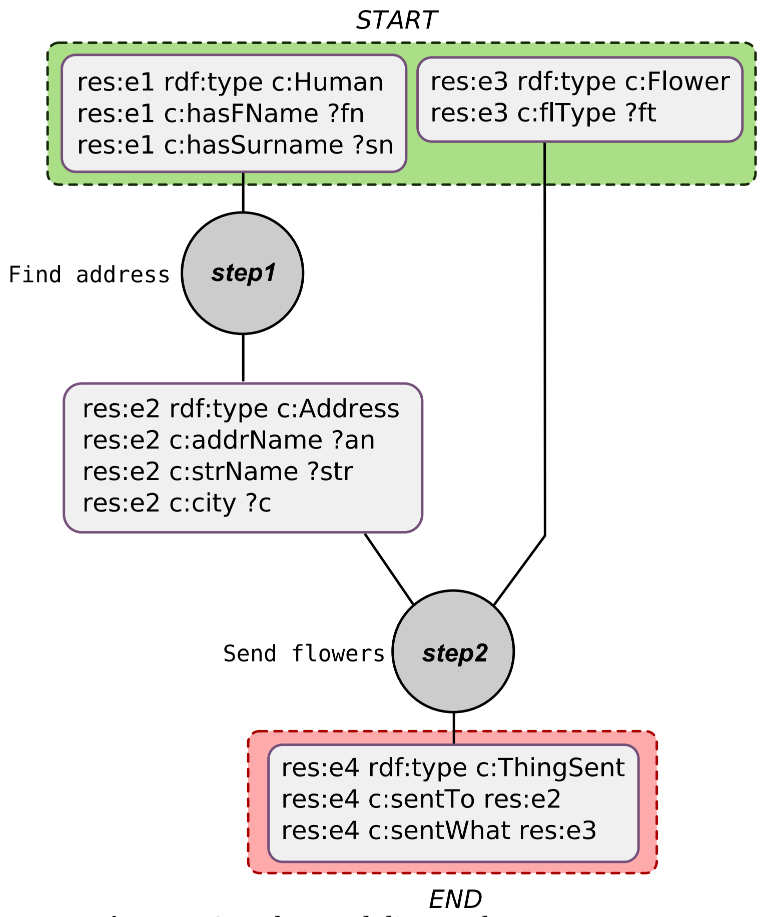

Figure 4.3 *- Flower delivery abstract process.*

4.3.2 Executable process

The difference between an abstract and an executable process is in the binding. Binding is a connection between an abstract step and a concrete Ontonut call with concrete parameter values. This binding depends on the context and user-specified goals. In this case we are sending a flower of type rose to person whose first name is John and surname is Doe. An example can be seen in Figure 4.4.

4.3.3 Process configuration ontology

Process configuration ontology is common for all types of processes. It deals with issues as bindings of variables, bindings of Ontonuts, user context and runtime self-configuration instructions.

Firstly, all resources that are used as an input to the process must be defined. In our case these resources are person name and surname together with the flower type. We have to define values for these variables, e.g. “John”, “Doe” and “rose”. Therefore the ontology must include variable-value binding vocabulary.

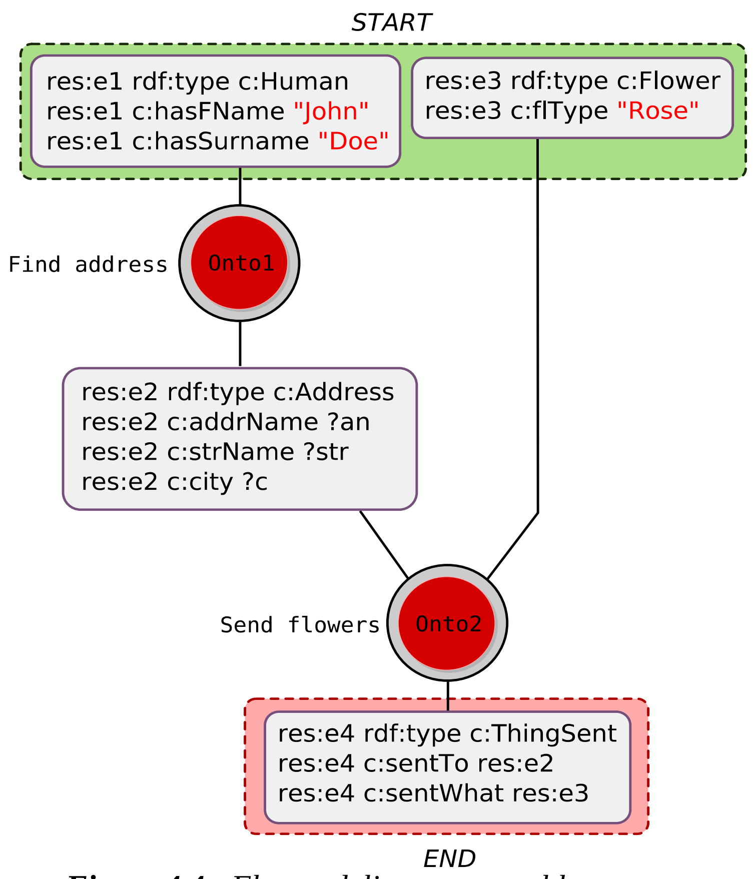

Figure 4.4 *- Flower delivery executable process.*

Secondly, all abstract steps must be bound to particular Ontonuts. Under this we understand that abstract process variables will be bound to real Ontonut inputs and outputs. Let's say that in our example we found an Ontonut that could implement the first abstract step. Let's assume that this Ontonut needs two inputs of type string – input A (first name) and input B (surname). In the abstract process these variables are named as `?fn` and `?sn`. The binding will then specify that the process variable `?fn` should be used as input A and the process variable `?sn` should be used as input B. The same happens to the outputs of Ontonuts. An example of a configuration for a single abstract step could be as following (only basic parameters are shown):

```
spel:abstrStep1 con:stepConfigured {
    con:ImplementedBy con:ontonut Onto1 .
    con:Inputs sapl:is {
        ont1:inputA con:boundToVar ?fn .
        ont1:inputB con:boundToVar ?sn
    } .
    con:Outputs sapl:is {
        ont1:outputA con:boundToVar ?nm .
        ont1:outputB con:boundToVar ?str .
        ont1:outputC con:boundToVar ?c .
    }
}
```

The third thing that must be included in the configuration is the minimal set of instructions necessary for runtime self-configuration. These instructions could involve director's name, contact address, contact method or context-specific information.

User context defines the circumstances of the process. It may affect only some steps or the whole process. From the ontological point of view, there are two kinds of beliefs in the user context. The first kind defines general attributes of the goal. These may include issues such as time (time critical vs. time non-critical process) and price. The second kind of beliefs describes the domain-specific issues. In our example the domain could be understood as flower sending. A domain-specific belief in this domain could be the fact that to living people you give an odd number of flowers. In this context the application will send only 1, 3, 5, … flowers.

In general the first kind of beliefs can be related to any process. Issues such as time and price are important in any kind of process. This implies that the ontology for defining these beliefs should be common for all the processes and therefore domain-independent. For this reason it should be a part of the configuration ontology.

4.3.4 The initial process configuration steps

Pre-configuration phase

During this phase an abstract process is chosen based on user's goal. The communication between the user and the system happens through the director agent. There are several ways how a director agent can be contacted. There can be a pool of director agents and one agent acting as an introducer. Every agent platform will have exactly one introducer agent. The introducer agent may run an HTTP server whose address will be publicly known. This address will act as the platform contact address. Every time a user wants to run a process on a particular platform, it will contact its introducer agent and based on the process category, the introducer agent will delegate him to the appropriate director agent. The user will communicate the goal with the director agent and from this point on the initial configuration will take place. The pre-configuration phase can be seen in Figure 4.5.

In the next step the user tells the director the desired goal. The task of the director is to choose the appropriate abstract process that would meet this goal. In our example the user's wish might be specified as “I want to send some flowers to some person”. This can be semantically annotated as following:

```
?x rdf:type c:ThingSent .
?x c:sentTo ?y .
?x c:sentWhat ?z .
?y rdf:type c:Human .
?z rdf:type c:Flower
```

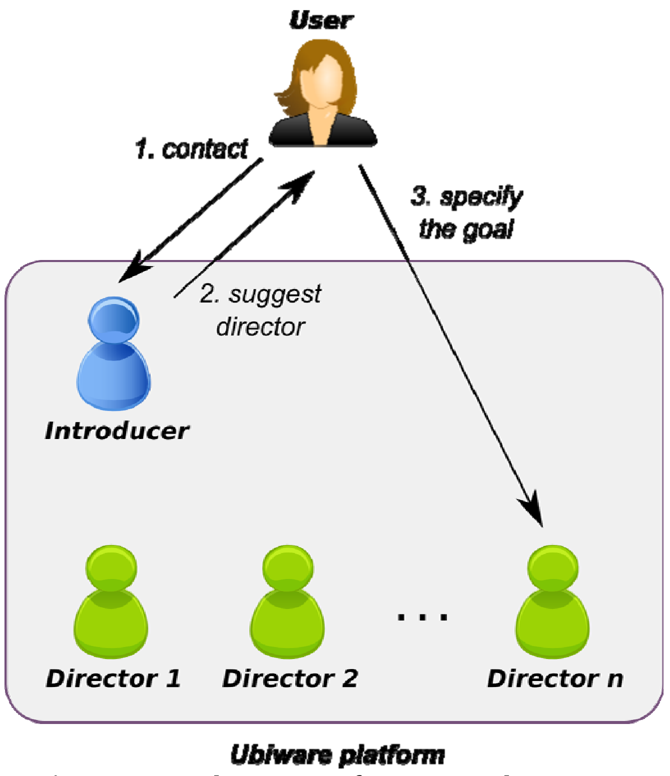

Figure 4.5 *- The pre-configuration phase steps.*

Along with this goal definition some context information may be provided. As mentioned earlier, issues as time or price are properties of any process. Let's say that the user in our example doesn't care about the price and she/he doesn't specify anything about the process being time-critical or not. Also, the user wishes to chose the process if several candidates are found. This could be specified as following:

```
con:process con:price con:priceUnbound
con:process con:chosenBy con:user
```

Clearly a broad definition of a goal will results in more abstract processes to be found. In our example the user wishes to choose the process that he or she thinks will fulfill his/her goal the best. The other option could be choosing the best process automatically based on some criteria (price, time, etc.).

After the user specifies the goal and optionally the context, the director agent will contact the abstract process repository and search for all the abstract processes that fulfill the goal specified by the user. An abstract process is chosen from the repository based on semantic match of the goal specified by the user and the goal of processes stored in the repository. Since in our example the user specified that she/he wants to choose the process, the director will offer several processes. Note that in this stage it is still just an abstract process with no bindings to real Ontonuts. The user chooses a particular process and the phase of initial configuration starts.

The initial configuration

The input to this stage is an abstract process chosen by the director agent, a goal and optionally a user-defined context. In this phase the director agent is trying to find the suitable Ontonuts and configure them. The first step is to look at the process and one-by-one replace the abstract steps with Ontonut calls. For every abstract step, the ODF is called with the following query:

```
I want { You answer {
  ?agent hasService {
    input is {...input patern...}
    output is {...output patern...}
    serviceURI is ?uri
}}
```

By knowing the values of variables `?agent` and `?uri`, the director knows exactly which Ontonut on which agent should be used in order to achieve this particular abstract step. In general, several results per one abstract step can be returned by ODF. In this case the director agent has to choose the most suitable Ontonut. This can be done based on the context the user provided. For instance, if the user considers the process to be time critical, the Ontonut with the lowest guaranteed time will be chosen. For this purpose it may be necessary to negotiate with agents having certain Ontonuts in order to obtain this information.

After substituting all the abstract steps with real Ontonuts and binding the process variables to actual inputs and output of Ontonuts, the Ontonut binding process is done. At this point the executable process is almost ready. The only thing missing is the initialization of starting parameters. For this reason the user is asked by the director agent to provide these parameters. In our case these parameters are person name, person surname and flower type. Now, the abstract process becomes an executable process.

4.4 Runtime self-configuration

4.4.1 Situation analysis

As mentioned earlier, the purpose of runtime self-configuration is to keep the process running and reaching its goal even if the environment changes. There can be three main types of events.

The first type is a case when a component (Ontonut) is missing. This situation occurs when an Ontonut chosen in the initial configuration phase is not available in the runtime. One of the reasons could be that the Ontonut is temporarily unavailable.

The second type of event is an Ontonut failure. This involves cases such as resource unavailability, missing property of a resource or just inability to perform the task. An example could be a case when the process uses a credit card resource and it will find out that the card expired and therefore it cannot proceed.

The third type of event occurs when there is a change in the user context. During the process execution, the user may change his/her original decisions. An example could be a case when the user decides that a process originally considered to be time non-critical will become a time critical one.

First two types are fundamentally different from the third one. In case of a missing Ontonut or a failure, the situation originates from the Ontonut. This means the source of the situation is in the process itself, whereas in case of context change the source of the situation is the user. If the user changes his/her mind and decides that the process context is different, then this information must be delivered top to bottom, from the user to the director and to particular agents running used Ontonuts. In worst case this may be understood as a change of the rules while the process is in progress and it may trigger a complete replanning. In our work we deal mostly with the first two cases.

4.4.2 Self-management vs. branching

At this point it is necessary to distinguish between a self-management functionality and normal functionality implemented by an Ontonut. As an example we can take an Ontonut implementing a payment for flowers (Figure 4.6). In the beginning the Ontonut tries to pay with a credit card of type Visa. If the transaction is unsuccessful, it will try MasterCard and if this one is unsuccessful as well, it will try Paypal account. Obviously, we can observe a behavior that could be classified as self-healing in broader sense, because it incorporates several backup solutions if the main branch fails. However, we don't believe that this is self-healing as it is understood in the vision of autonomic computing (Kephart and Chess, 2003). Self-healing takes place when the operation fails and the component (in our case Ontonut) is not able to handle this situation. In our example this would happen if also the last option (paypal) fails. In this case the component is “clueless” and self-management must take place.

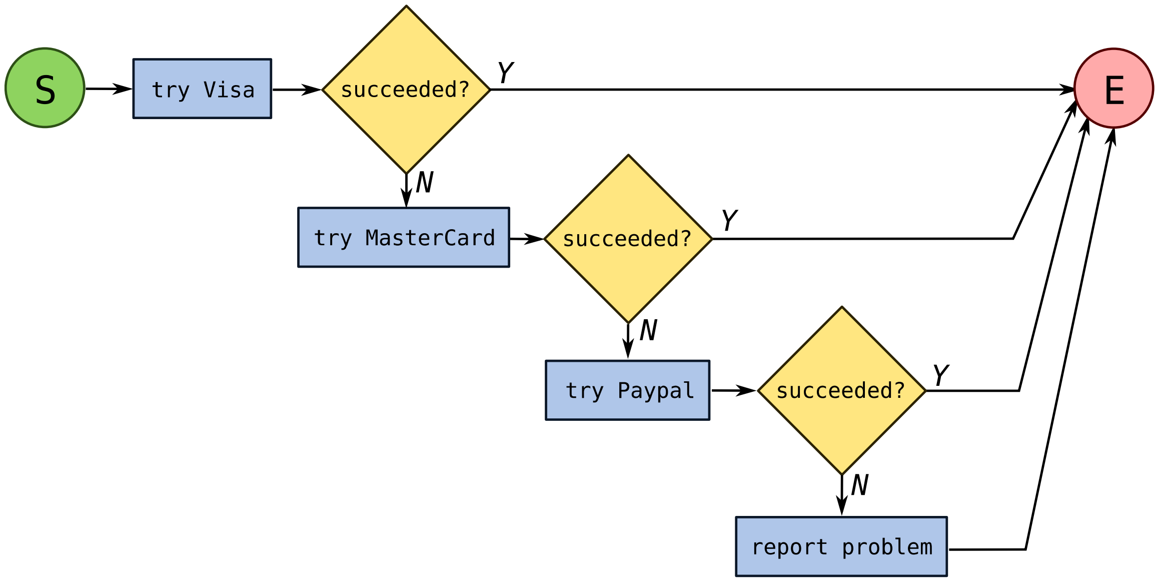

Figure 4.6 *- Branching example.*

4.4.3 Architecture

As a basis for our architecture we took a three layer model proposed by Kramer and Magee (Kramer and Magee, 2007). This model originates from a model described by Gat (Gat, 1997). Gat's model was originally used in robotics and it consisted of the following layers. First layer called Control was dealing with reactive feedback control. The second layer was called Sequencing and it' purpose was to react to the changes and execute plans. The third layer was Deliberation layer and its task was planning.

Kramer and Magee used Gat's model to derive a model suitable for self-configuration in component-based architectures. The new model consists of three layers, same as Gat's (Figure 4.7). The first layer is called Component control. It contains self-tuning algorithms, sensors and actuators. Its goal is to adjust the parameters of the component and if a situation occurs that the component cannot deal with, it will be reported to a higher layer. The second layer is called Change management and its task is reactive plan execution. This layer reacts to changes reported by the lower layer and executes a pre-specified plan that will handle the change. As the result a new component may be chosen, an old component may be reconfigured or connections between components can be changed. The last, third, layer is called Goal management and its task is to plan. If the second layer cannot find a suitable plan to handle the change reported by the first layer, it will contact the third layer and ask it for replanning.

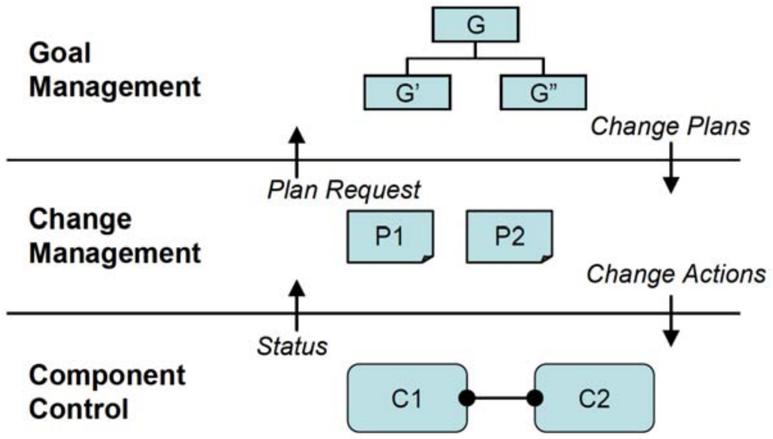

***Figure 4.7** - Three layer architecture as proposed by* (Kramer and Magee, 2007).

In case of Ubiware architecture we decided to use a similar approach. The Component control layer functionality is implemented in agents running particular Ontonuts. They monitor the Ontonut execution and if there is an exceptional situation that the Ontonut cannot handle, it is reported to the second layer. Monitoring is based on a control loop (Shaw, 1995).

The second layer (Change management) is implemented in the director agent. When it receives a request from an agent implementing a certain Ontonut, the director agent can do the following actions. Firstly, if the user wanted to be contacted in case of a fault, he or she will be contacted with detailed information about what happened. The user may decide if a new component (Ontonut) should be found (e.g. different flower payment service) or if he/she will provide more information for the failed Ontonut (e.g. a different type of credit card that would be accepted). Since self-management is trying to minimize the contact with user, this option is left open just in case of processes that user wants to have bigger control over. Secondly, if the user didn't want to be contacted, a replacement will be found by the director agent. The replacement is being found in the same way as in the initial configuration phase. The ODF is contacted and a suitable Ontonut is chosen (Figure 4.8). If no Ontonut can be chosen, then the third layer is notified.

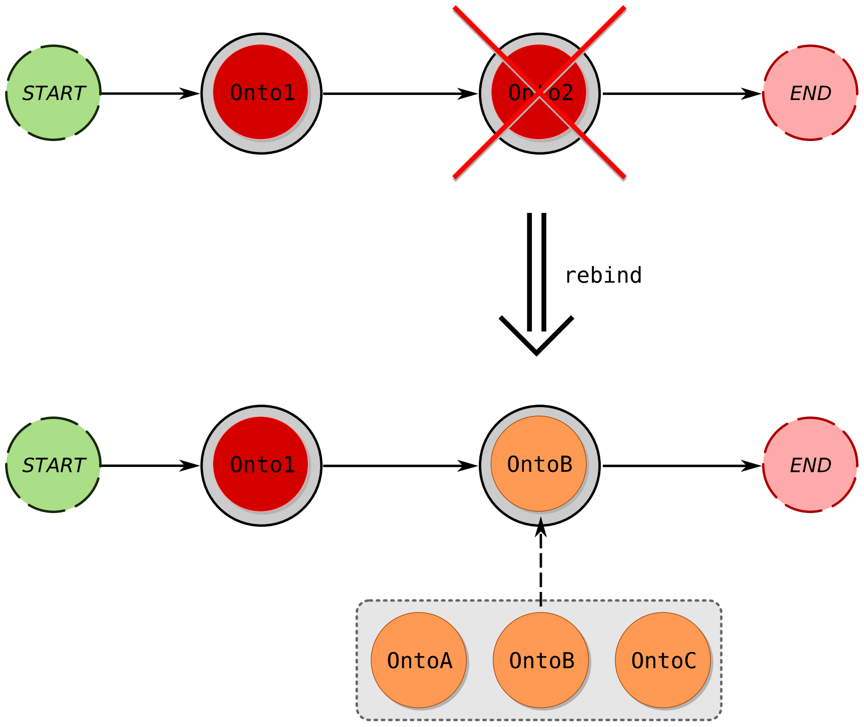

Figure 4.8 *- Rebinding of Ontonut OntoB instead of the failed Onto2.*

The third layer (Goal management) can be implemented in the director agent or it can be implemented in an external service. This layer involves planning which is in general time consuming. If there is no Ontonut that could implement the desired functionality (abstract step), the process must be replanned. If there is a possibility to reach the final goal from the previous step using some other combination of Ontonuts, then this new path will be taken. If there is no way to reach the final goal, then it is reported to the user as a process failure and the user must decide what should be done next. In our flower delivery example the replanning could occur if the second step (flower delivery) cannot be completed and no other suitable Ontonut is found. In this case the process might be replanned and two steps might be necessary in order to achieve the final goal (Figure 4.9). These two steps could be flower order, where the flowers are ordered, but not paid for, and bank transfer, where a payment through e-banking will take place.

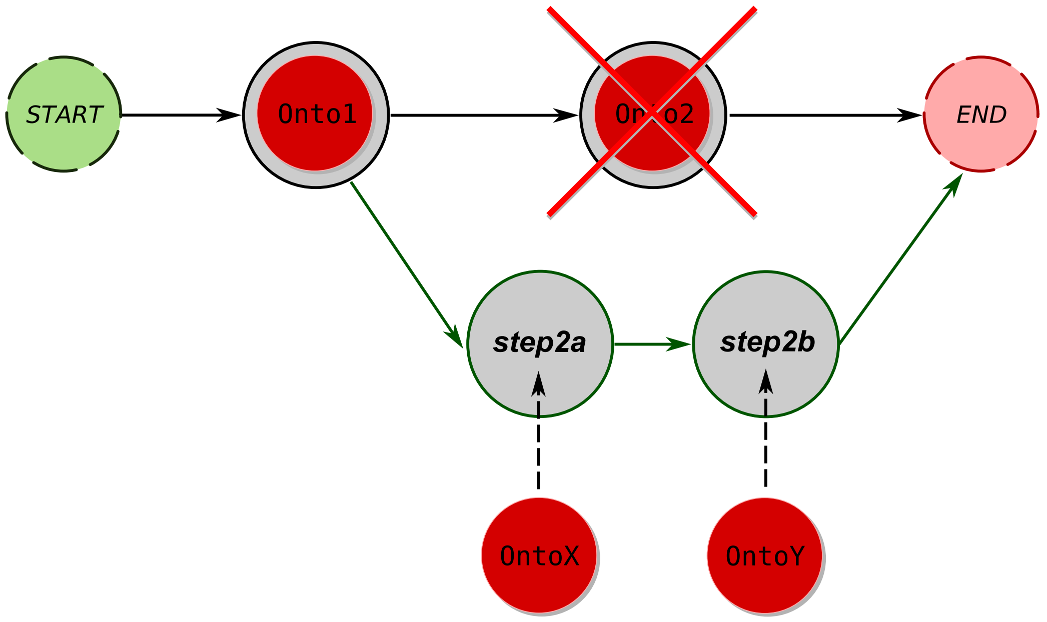

***Figure 4.9** - Replanning due to Ontonut Onto2 failure.*

4.5 Open questions

In this section we will sum up open questions related both to initial configuration and runtime self-management.

4.5.1 Ontonut binding

This phase takes place when we want to find an implementation for every abstract step in the abstract process. Since every abstract step has a semantic description of expected inputs and expected outputs, it is possible to find a relevant Ontonut. In our initial example (Figure 4.3) we can see that for the second abstract step we need an address description and a flower description. The ideal situation that can happen during the binding process is that we will find an Ontonut that will produce the exactly same kind of output with exactly same kind inputs. In other words, the ideal situation is when we find a “perfect” Ontonut that needs just an address and flower type and it can order a flower and send it to this address, exactly as we expect in the abstract step.

However, a situation may arise when we don't find a “perfect” Ontonut. We might find an Ontonut that can send flowers to some address, but as an input it needs an address, flower type and also credit card information. In this case the question is: Should we use this Ontonut and consider the abstract step bound to it? This is rather a philosophical question.

If we decide to bind this Ontonut, we may say that it is not the original abstract process and thus we are creating a new abstract plan/process. On the other hand, the credit card information can be considered just another input, same as name, surname or flower type (Figure 4.10).

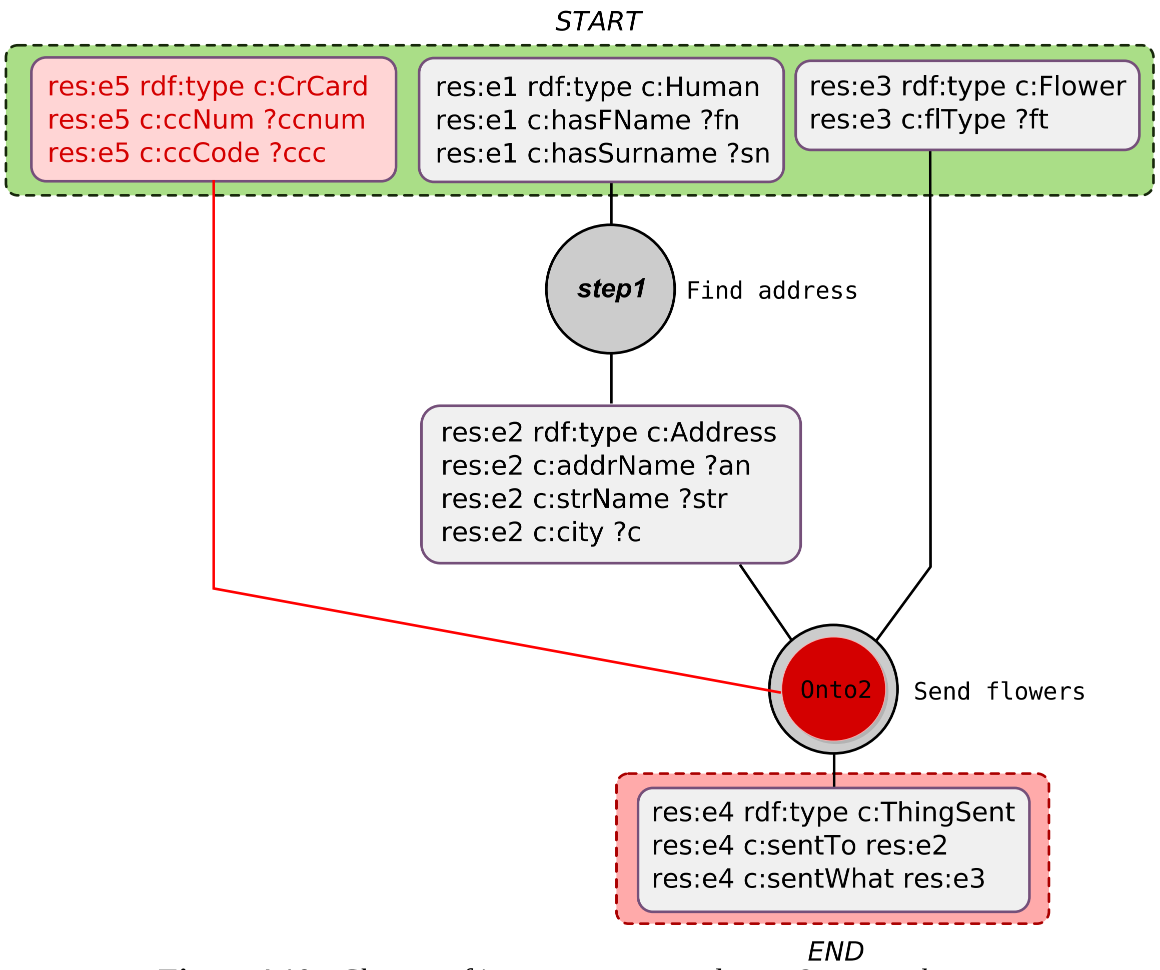

Figure 4.10 *- Change of input parameters due to Ontonut change.*

4.5.2 Ontonut's error reporting capability

As mentioned earlier, we expect that in case of an error every Ontonut has the ability to end its execution gracefully and report an error. This assumption may be naive, since Ontonuts are written by different vendors. There should be some mechanism that forces the Ontonut creators to add this functionality. For example in object-oriented languages there is a concept of interface. If the architecture forces the creator to implement a certain interface in his/her class, all the methods of that interface have to be implemented and thus the architecture is always sure that they will be available.

4.5.3 Change management and partial plans

The task of the second layer of self-management architecture is reactive plan execution. We limited it to Ontonut rebinding. If the current Ontonut is incapable of performing the functionality required in that particular abstract step, then another Ontonut is chosen. At this point the question is: Could we have some partial plans prepared that could act as an Ontonut replacement? This is again a philosophical question.

Let's imagine that we have an abstract step whose goal is to book a family holiday. This step is just a part of some bigger process. Let's assume that we have an Ontonut that is able to perform this step. Let's assume that during the initial configuration we decided to bind the abstract step exactly to this Ontonut, but later in the process we will find out that it is not available (Figure 4.11). Obviously this is an issue for the Change management layer. At this point, the Change management layer is trying to find a suitable replacement in a form of an Ontonut. If it doesn't find it, the whole issue is reported to higher, third, layer. Let's assume that the third layer will find an alternate plan using two separate steps - flight booking (step2a) and hotel booking (step2b).

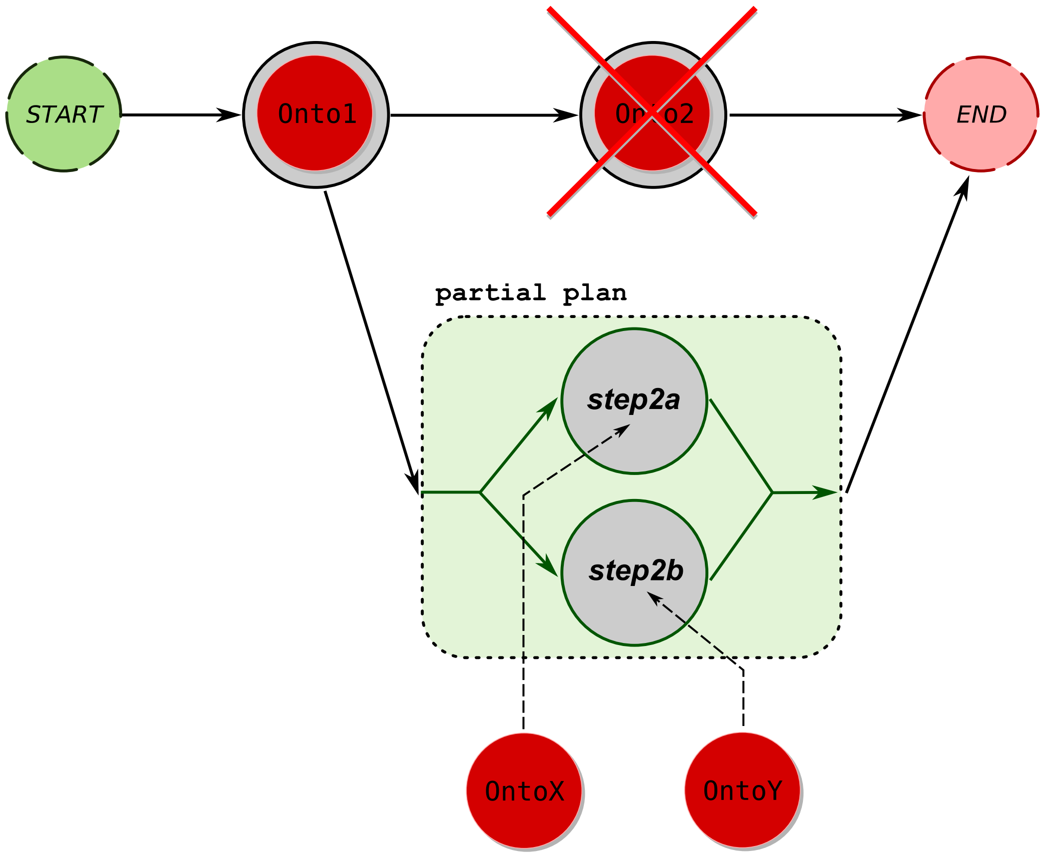

Figure 4.11 - *Replacement of an Ontonut by a partial plan.*

The original question asks if we should take into consideration also partial plans when the second layer is looking for an Ontonut replacement. In other words, should the second layer also look for partial plans (e.g. booking a flight and booking a hotel) that could be used instead of the failed Ontonut? On one hand this may be considered replanning, which is the task of the third layer. On the other hand it involves just an insertion of a prepared partial plan, which is equal to Ontonut replacement.

4.6 Related work

Garlan and Schmerl (Garlan and Schmerl, 2002) propose externalization of the adaptation of a system. There have been many "internal" mechanisms to provide self-adaptation. Usually they had the form of programming language features (e.g. exceptions and run-time assertion checking) and algorithms (network protocols, self-stabilizing algorithms). They were able to discover a failure, but they were not well-suited for discovery of smaller changes, such as gradual degradation of performance. The other problem was that they were part of the application code and thus difficult to change during the runtime. As a solution they propose a model-based adaptation where the behavior of the system is monitored by components outside of the running system (Figure 4.12). They store a system model, which an abstraction of the running system. These monitoring components are also able to reason about the system based on the system model and information gathered during monitoring.

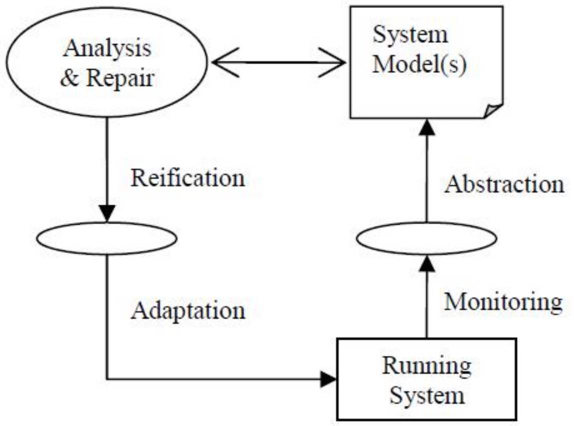

Figure 4.12 *- Model-based adaptation as defined by (Garlan and Schmerl, 2002).*

In their paper, Dobson et al. (Dobson et al., 2006) describe the field of autonomic communication. They also compare autonomic computing to autonomic communication. Several similarities can be found. Autonomic communication and autonomic computing have much in common. They both use decentralized algorithms and control and context-awareness. Both of them can be considered novel programming paradigms which are trying to achieve self-* properties. The goal in both cases is to simplify the management of complex structures and reduce the need for manual intervention and management. However, there are differences as well. Autonomic communication is more oriented towards distributed systems and services. A typical application could be the management of network resources at both the infrastructure and the user levels (Quitadamo and Zambonelli, 2007). On the other hand, autonomic computing is dealing more with application software and management of computing resources.

As mentioned earlier, Kramer and Magee (Kramer and Magee, 2007) describe a three layer architecture for self-managed systems. As benefits of architectural approach they consider generality, level of abstraction, good potential for scalability, good potential for an integrated approach and the amount of research done in the field of architectural approaches. Later they describe the proposed model and at the end they discuss several research challenges. A challenge for the first layer (Component layer) is the ability to preserve the information in case a component changes. A component change should not cause any information loss or unwanted behavior. On the Change management layer they identified the issue of distribution and decentralization. They consider it the biggest difference between the problem of self-management of complex systems and robotics, where their architecture originates from. As an unsuccessful experiment they mention a case when they were trying to design a system, whose management layer was not implemented centrally, but in each component separately. If this kind of system was supposed to work, it would need to have a total order reliable broadcast bus. This is useful information in context of Ubiware development. As a challenge for the third layer they identify the problem of high-level goal specification. They express the need for goal specification readable by humans and understandable by machines. In Ubiware we deal with this issue and semantic web could be one of possible answers to this problem.

4.7 Conclusions

In this workpackage we discussed issues involving initial process configuration and runtime self-management. In the beginning we defined terms like abstract process, executable process, abstract step, binding, etc.

We proposed an architecture for initial configuration involving Abstract process repository, director agent, ODF and several Ontonut implementations. We described the initial process configuration phase, in which an executable process is built based on an abstract process. This phase involves Ontonut binding and variable binding.

Later, we discussed the issue of self-management in runtime. We used a three layer self-management architecture proposed by Kramer and Magee (Kramer and Magee, 2007) and adapted it to suit our needs. The first layer of the architecture is implemented in agents running particular Ontonuts. They monitor the Ontonut execution and if there is an exceptional situation that the Ontonut cannot handle, it is reported to the second layer. The second layer is implemented in the director agent and its goal is to deal with situations that were not handled by the first layer. Depending on the user context, the second layer will try to use an alternate Ontonut or contact the user to ask for additional instructions. As an open philosophical question we leave the issue of replacing a single Ontonut with prepared partial plans. Another open question is the issue of binding Ontonuts whose precondition is broader than the precondition of the Ontonut being replaced. The third layer is mostly dealing with planning. In case the second layer fails to resolve the problem, the third layer will have to replan and reach the original goal using another set of Ontonuts.

At the end we discussed related work in the field of self-management and its impact on Ubiware.

UBIWARE Deliverable D3.1:
Workpackage WP5:
Task T3.1_w5:
Workpackage leader: *Oleksiy Khriyenko*

5 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.

According to the last discussion during the steering group meeting, we agreed to concentrate 3rd year project research on business issues, commercialization steps of the results. Now when we have clear vision of the idea, have the elaborated prototype of an initial idea, we are ready for the next valuable step during WP5's Year 3. This step will consist of two parts: elaboration of the general architecture of the product (necessary components, tools and utilization models) and commercialization part (business and market analysis, business models, promotion, distribution and etc.).

During WP5's Year 3 (the *Commercialization* phase), therefore, the following research questions are to be answered:

- What should be the general architecture of the product – 4I(FOR EYE) tool package so that it will be possible to build and further extend a different services based on the product? What are the requirements for the product, for the product components, what are the necessary modifications and the use cases of the product utilization?
- What are the commercialization and marketing steps?

5.1 4I(FOR EYE) Browser

4I(FOR EYE) Browser is a context sensitive visual resource browser. The technology used behind the browser (Khriyenko, 2007a) enables creation of a smart human interface through flexible collaboration of an Intelligent GUI Shell, various visualization modules, which we refer to as MetaProvider-services, and the resources of interest. Semantically enhanced context-dependent multidimensional resource visualization (Khriyenko, 2007b) provides an opportunity to create intelligent visual interface that presents relevant information in more suitable and personalized for user form. Context-awareness and intelligence of such interface brings a new feature that gives a possibility for user to get not just raw data, but required integrated information based on specified context. Ability of the system to perform semantically enhanced resource search/browsing based on resource semantic description brings a valuable benefit for today Web and for the Web of the future with unlimited amount of resources.

5.1.1 General architecture of the browser

5.1.1.1 GUI-Shell

GUI-Shell has client-server architecture and represented by frontend web-page (HTML, JavaScript) and backend server part (ApacheTomcat, Java) of the system.

Resource repository format convertor. Through this module user selects appropriate repository (source file) with resource descriptions to be a basis for the system. Module converts original format of a repository to the internal format of the system.

Resource search system allows user to perform a resource search process. There are two types of input data (presented as a set of keywords) that describe type of a resource and resource content. Result of the search is ranked set of resources.

User profile browser/editor reads default settings from user profile in case when system cannot get an optimal decision what visualization context should be applied for selected resource, which visualization module choose for visualization and etc. This default information is used to make browser more reactive, dynamic and user friendly. Usually user profile is updated automatically by system that applies some learning techniques, but also can be modified by user through separate interface (current prototype does not have such an option).

Visualization contexts description browser/editor is aimed to browse and update (create and change) Visualization contexts description repository. User can create a new instance of existing visualization context description by customization of changeable parameters. Adding and creation of the editing pages of the new Visualization contexts should be perform by experts via adding changes to the browser. Some more flexible methods of adding new Visualization contexts that do not need any changes in the browser could be considered and elaborated.

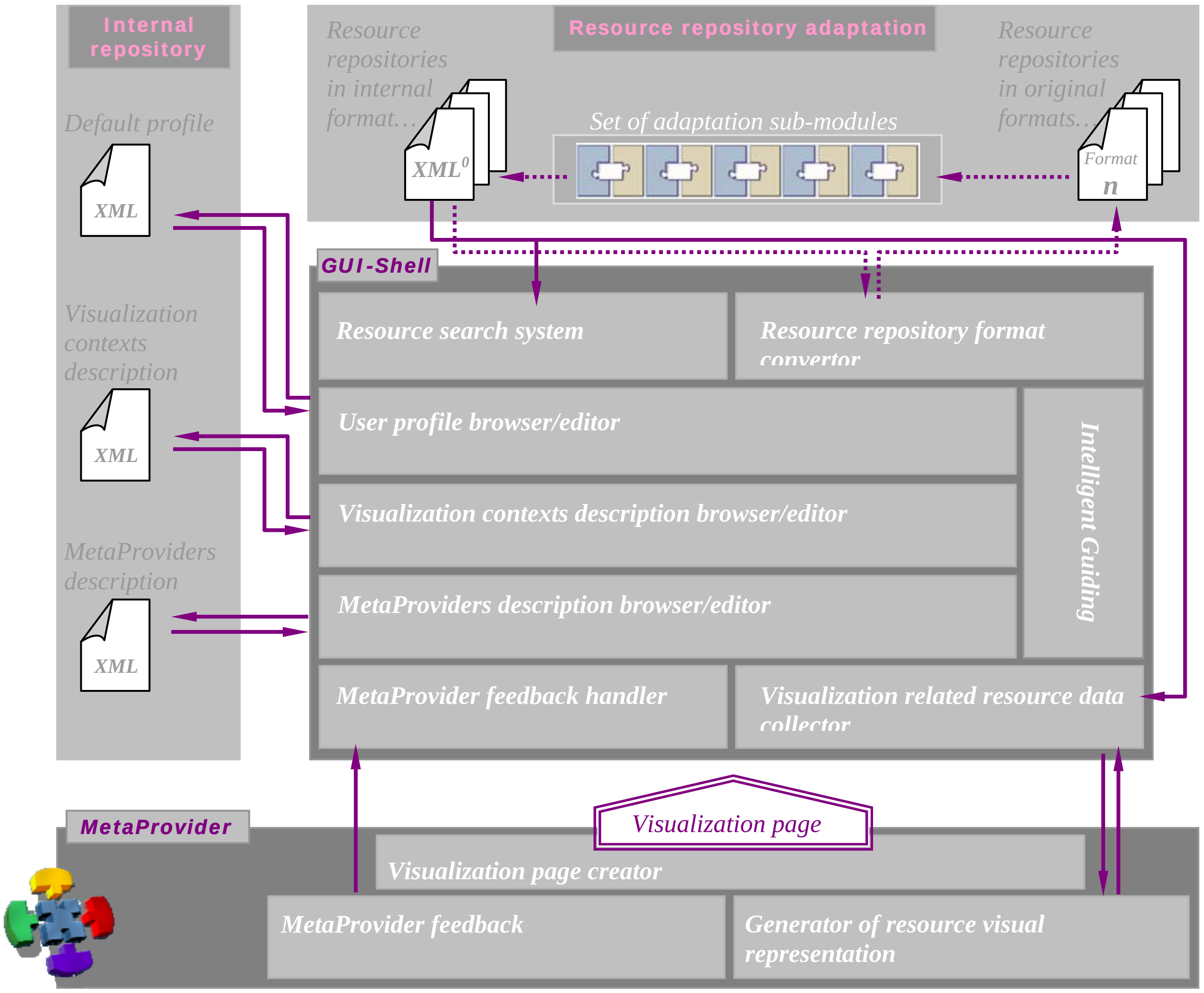
Internal repository
Default profile
XML
Visualization contexts description
XML
MetaProviders description
XML
Resource repositories in internal format…
XML0
Resource repository adaptation
Set of adaptation sub-modules
Resource repositories in original formats…
Format n
GUI-Shell
Resource search system
Resource repository format convertor
User profile browser/editor
Intelligent Guiding
Visualization contexts description browser/editor
MetaProviders description browser/editor
MetaProvider feedback handler
Visualization related resource data collector
MetaProvider
Visualization page
Visualization page creator
MetaProvider feedback
Generator of resource visual representation

Figure 5.1 *- General architecture of the browser.*

MetaProviders description browser/editor. This module communicates with *MetaProviders description repository* and is used for adding new MetaProvider to the system as well as editing these descriptions.

Visualization related resource data collector. This module collects all necessary for MetaProvider information related to resource visualization, based on requirements described in MetaProvider description, and sands collected data with a request for visualization to correspondent MetaProvider. In more advance case, this module communicates with MetaProvider and helping it to find necessary data.

Intelligent Guiding module. One of the intelligent techniques of 4I Browser is a technique for automatic dynamic selection of a visualization context. The logic is based on the user browsing history (browsing route - a sequence of visualized resources and correspondent visualization contexts) and the experience of other users (history of browsing routes). 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, browser becomes a smart search system that leads the user in proper direction. The module keeps 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 module is a list of sorted visualization contexts in a context of browsing history and experience of predecessors. 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 (using the *Default profile* of the Browser).

MetaProvider feedback handler. To keep communication with user in interactive way, each visualization module should provide a feedback function that returns ID of pointed by user resource to GUI-Shell. This information is used by GUI-Shell to present descriptive information of the resource to the user.

Resource repository adaptation. Current prototype of 4I Browser uses own internal XML-based format for data representation. Browser should perform adaptation of different data representation formats into internal format, to be used in various systems as an integrated module. Thus such module is a set of different adaptation modules that will be used depending on a format of a raw input data.

Internal repository: Internal repository is presented by files that contain default user profile(s) of the browser, specification of resource visualization contexts and descriptions of connected and used visualization modules (MetaProviders).

- *Default profile:* This file contains initial default settings of the browser. Current prototype contains just settings related to information that allows browser to interact with a user in more dynamic way. There is information about what visualization context should by applied for certain class of the resource and which MetaProvider should be used to visualize certain resource in certain context if Intelligent Guiding module of the browser did not provide appropriate information.

defaultProfile.xml – default profile of the browser (example extraction).

```
<profile>
<contexts>
        <context forResource="Groups">Context_1</context>
        <context forResource="Project">Context_3</context>
        <context forResource="Organization">Context_2</context>
        <context forResource="Researcher">Context_5</context>
        <context forResource="Worker">Context_5</context>
        <context forResource="Idea">Context_6_1</context>
        <context forResource="Proposal">Context_6_1</context>
        …
</contexts>
<metaProviders>
```

```
        <metaProvider forResource="Groups" forContext="Context_1">mp_1</metaProvider>
        <metaProvider forResource="Groups" forContext="Context_2">mp_1</metaProvider>
        <metaProvider forResource="Project" forContext="Context_3">mp_1</metaProvider>
        <metaProvider forResource="Project" forContext="Context_4">mp_1</metaProvider>
        <metaProvider forResource="Organization" forContext="Context_2">mp_1</metaProvider>
        <metaProvider forResource="Researcher" forContext="Context_5">mp_1</metaProvider>
        <metaProvider forResource="Worker" forContext="Context_5">mp_1</metaProvider>
        <metaProvider forResource="Idea" forContext="Context_6_1">mp_2</metaProvider>
        <metaProvider forResource="Proposal" forContext="Context_6_1">mp_2</metaProvider>
        <metaProvider forResource="Idea" forContext="Context_6_2">mp_2</metaProvider>
        <metaProvider forResource="Proposal" forContext="Context_6_2">mp_2</metaProvider>
        …
</metaProviders>
</profile>
```

- *Visualization contexts description:* This file contains descriptions of the visualization contexts used by the system. There are information related to the context’s attributes and definitions for which classes of the resource each visualization context should be used. In current prototype we consider two different types of contexts. There are contexts that are not accompanied with a specific context related data and just name the context (to give it semantics), and those, which have certain personalized data to be used by visualization modules (MetaProviders) as an input data. Such more personalized context descriptions can be considered as a detailed sub-context’s descriptions.

contexts.xml – visualization contexts description – first type (example extraction).

```
<contexts>
        <context>
                <contId>Context_1</contId>
                <name>personnel, staff ...</name>
                <forClasses>
                        <class>Groups</class>
                        <class>Class_2</class>
                        <class>Class_3</class>
                </forClasses>
        </context>
        <context>
                <contId>Context_2</contId>
                <name>related projects</name>
                <forClasses>
                        <class>Groups</class>
                        <class>Organization</class>
                        <class>Class_3</class>
                </forClasses>
        </context>
        …
</contexts>
```

In current prototype we use **resClosenessContext.xml** file to locate a second type context description related to resource closeness visualization context, where we store specific for each sub-context data.

resClosenessContext.xml – visualization contexts description – second type (example extraction).

```
<closenessContexts>
        <closenessContext>
                <closenessContext_id>Context_6_1</closenessContext_id>
                <closenessContext_name>resource closeness 1</closenessContext_name>
                <calculation_method>GDCM_WithAverageCorrection</calculation_method>
                <calculation_methods>
                        <value>GDCM_WithoutCorrection</value>
                        <value>GDCM_WithAverageCorrection</value>
                        <value>GDCM_WithMedianCorrection</value>
                </calculation_methods>
                <fieldContext>
                        <field_type>textField</field_type>
                        <field_property_id>ideaField_1</field_property_id>
                        <field_property_name>Idea field 1</field_property_name>
                        <field_significance>0.1</field_significance>
                </fieldContext>
                <fieldContext>
                        <field_type>keyWordsField</field_type>
                        <field_property_id>ideaField_2</field_property_id>
                        <field_property_name>Idea field 2</field_property_name>
                        <field_significance>0.3</field_significance>
                </fieldContext>
                <fieldContext>
                        <field_type>complexTextField</field_type>
                        <field_property_id>ideaField_3</field_property_id>
                        <field_property_name>Idea field 3</field_property_name>
                        <field_significance>0.3</field_significance>
                        <subprop_names>
                                <subprop_name>subProp 1</subprop_name>
                                <subprop_name>subProp 2</subprop_name>
                                <subprop_name>subProp 3</subprop_name>
                        </subprop_names>
                        <corClasses>
                                <corClass>
                                        <class_significance>0.3</class_significance>
                                        <value>a1</value>
                                        <value>a2</value>
                                        <value>a3</value>
                                        <value>a4</value>
                                        <value>a5</value>
                                        <value>a6</value>
                                </corClass>
                                <corClass>
```

```
                                        <class_significance>0.3</class_significance>
                                        <value>b1</value>
                                        <value>b2</value>
                                        <value>b3</value>
                                        <value>b4</value>
                                        <value>b5</value>
                              </corClass>
                              <corClass>
                                        <class_significance>0.4</class_significance>
                                        <value>c1</value>
                                        <value>c2</value>
                                        <value>c3</value>
                              </corClass>
                    </corClasses>
          </fieldContext>
          <fieldContext>
                    <field_type>numberField</field_type>
                    <field_property_id>ideaField_4</field_property_id>
                    <field_property_name>Idea field 4</field_property_name>
                    <field_significance>0.1</field_significance>
          </fieldContext>
          <fieldContext>
                    <field_type>intervalField</field_type>
                    <field_property_id>ideaField_5</field_property_id>
                    <field_property_name>Idea field 5</field_property_name>
                    <field_significance>0.2</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>0.7</value>
                              <value>0.3</value>
                    </subField_significances>
                    <field_calculation_method>IFCM_2WithAverageCorrection
                                                  </field_calculation_method>
                    <field_calculation_methods>
                              <value>IFCM_1</value>
                              <value>IFCM_2WithAverageCorrection</value>
                              <value>IFCM_2WithMedianCorrection</value>
                    </field_calculation_methods>
          </fieldContext>
</closenessContext>
…
< closenessContexts>
```

- *MetaProviders description:* This is a file-storage for the registered visualization modules (MetaProviders). Mainly description of the module contains definition of input and output data for visualization of certain class of resources in certain context. This information is used by GUI-Shell of the browser to collect necessary data before MetaProvider invocation.

metaproviders.xml – MetaProvider's description (example extraction).

```
<mps>
	<mp>
		<mpId>mp_1</mpId>
		<name>MemberOf visualizer</name>
		<link>http://localhost:8080/MP_GroupResourceVis/visualization.jsp</link>
		<listener>http://localhost:8080/MP_GroupResourceVis/MP_Listener</listener>
		<rescont>
			<resClass>Groups</resClass>
			<cont>
				<contId>Context_1</contId>
				<_in>resId</_in>
				<_in>resLogo</_in>
				<_in>resLogo_w</_in>
				<_in>resLogo_h</_in>
				<_ins refProp="memberIs">
					<_in>resId</_in>
					<_in>resPhoto</_in>
					<_in>resPhoto_w</_in>
					<_in>resPhoto_h</_in>
				</_ins>
				<_out>resId</_out>
			</cont>
			<cont>
				<contId>Context_2</contId>
				<_in>resId</_in>
				<_in>resLogo</_in>
				<_in>resLogo_w</_in>
				<_in>resLogo_h</_in>
				<_ins refProp="projectIs">
					<_in>resId</_in>
					<_in>resLogo</_in>
					<_in>resLogo_w</_in>
					<_in>resLogo_h</_in>
				</_ins>
				<_out>resId</_out>
			 </cont>
			…
		</rescont>
		…
	</mp>
	…
</mps>
```

5.1.1.2 MetaProvider

MetaProvider is a kind of a service that has client-server architecture and represented by frontend JSP page and backend server part (ApacheTomcat, Java). The main duty of MetaProvider is to collect necessary data and create interactive visual model of contextual resource representation. It can utilize any web-based technologies to elaborate a model to be competitive among huge amount different MetaProviders provided by any third party in open 4I Environment.

Generator of resource visual representation. Here we can consider two behaviours of such a module. In simple case, it just gets all needed data from GUI-Shell and creates an appropriate visualization model. In more complex case, it requests data from GUI-Shell and collects necessary data from other various sources as well.

Visualization page creator. The result of MetaProvider work is a JSP page sent by MetaProvider to GIU-Shell with imbedded visualization model created by MetaProvider. MetaProvider can use different web-based technologies to produce resource visualization model. In current prototype we use X3D (and related) technology to present users interactive and dynamic 3D model of resource.

MetaProvider feedback. To fit one of the requirements for MetaProviders and provide interoperability of the browser, we have to implement feedback function when we develop visualization model. MetaProvider should provide the ID of selected/pointed by user resource (object) back to the GUI-Shell (*MetaProvider feedback handler*).

5.1.2 From prototype towards a product

Usually during the prototype development, developers are concentrated on particular domain, cover certain area and tasks. To become a product, prototype should pass through a number of stages that bring generalization to the prototype (ability to be used in different domains and arias), provide common information infrastructure and user/programming interfaces of the system to allow extension and configuration of it. Thus, we will try to discuss some of the challenges that we might face.

5.1.2.1 Common vocabulary

Considering 4I Browser as an open configurable system that can be constructed following module based approach, we face a challenge to provide a common ontology/vocabulary. This is a common problem for every information integration technologies nowadays.

Let us consider a case of local use of the system within one organization or entity where whole information infrastructure is concentrated in one hands and there is no interoperation with outside systems. If all the parts of the system (MetaProviders, context and MetaProviders descriptions and their editors, resource annotations, etc.) are developed and will be developed in future by the same group of specialists, then we can talk about a local centralized ontology/vocabulary. In this case, there are should not be any problems, because ontology creation will be done step by step when the new tasks will appear. And simplicity of such ontology extension process will depend on how well the expert knows the problem domain and correctness of the chosen approach of ontology creation.

Another story if we are talking about the case of really open environment where different parts of the system are developed and provided by different parties. And now we definitely have to have a common ontology/vocabulary and all these parties should follow the same common principles of development. Again, to create one big ontology that covers everything in one time would be the best way to do any system of information integration and interoperability, but unfortunately this is not realistic way. As in the previous case, process of ontology/vocabulary creation will be performed step by step by community of all players. And to make whole system reliable, this process should be controlled by correspondent entity (provider of the system). Any way we could not avoid appearance of sub-systems (former local systems) that would like to interact with each other and provide its knowledge and experience as external services. Then we will face a need of adaptation and bridging of different ontologies. This is one of the hot research topics in the Semantic Web area last years and some of the solutions are presented here: (Pinto and Martins, 2001), (Keet, 2004), (Alasoud et. al., 2005).

Current prototype does not have separate ontology/vocabulary as such, and uses some limited (task specific) imbedded one. Thus, ontology/vocabulary creation and manipulation mechanism is one of the challenges that has to be taken into account.

5.1.2.2 Source data adaptation

Source data adaptation is on of the challenges that we have to solve to make 4I Browser more or less working system. 4I Browser is kind of engine that provide context-sensitive visualization of resources via MetaProviders, it provides interoperability between different resources and services and adds some additional functionality. So, repository of resource descriptions is an input data for the Browser. Current prototype has imbedded sample repository of resource descriptions already in internal format. To make it able to base one any external repository, we have to elaborate general adapter that enable to convert data from any format to the required one. New data formats will appear all the time and will require new adaptation modules. Thus, optimal way for such module elaboration is to make it extendible, be able to add new adaptation sub-module for new data format transformation.

GUI-Shell should provide user interface to select external repository to be imported via appropriate adaptation sub-module from a list of available in Browser as well as interface for search of new sub-modules and their imbedding (Figure 5.2). To avoid unnecessary adaptations, cashing mechanism for adapted repositories can be realized in the Browser.

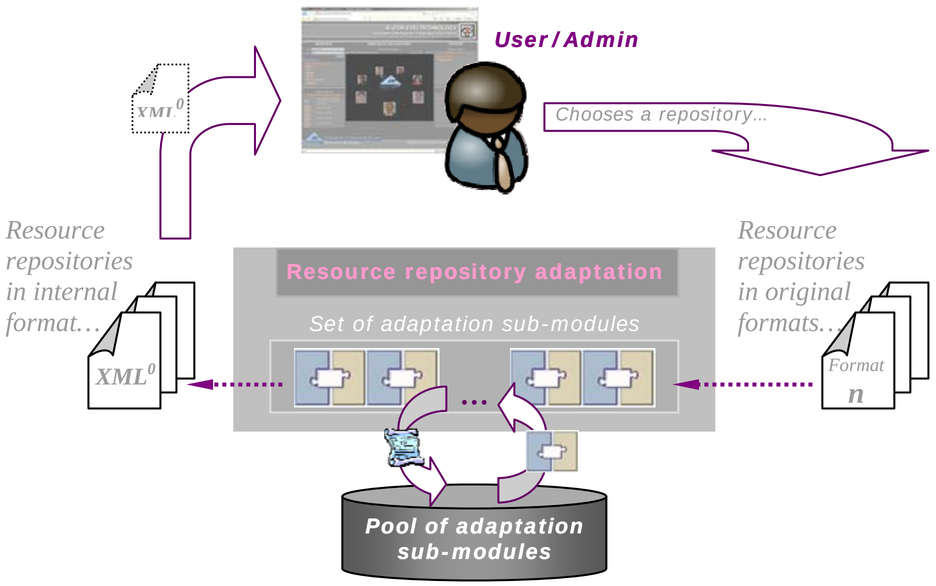
User / Admin
XML0
Chooses a repository...
Resource repositories in internal format…
Resource repositories in original formats…
Resource repository adaptation
Set of adaptation sub-modules
XML0
Format n
Pool of adaptation sub-modules

Figure 5.2 *– Source data adaptation.*

Regarding to the plan for Inno-W industrial case, we are going to elaborate adaptation sub-module to convert their RDF repository into internal format. It will be just one adaptation sub-module, but we are going to implement general functionality of the Browser according to modular approach of source data adaptation.

5.1.2.3 Interfaces for manipulations with context and MetaProvider descriptions

We consider 4I Browser as an open environment with a huge amount of different visualization contexts and even bigger amount resource visualization modules. There is no need to keep information/description about all of them inside the GUI-Shell. Moreover new context and MetaProviders will appear with a time. Thus, local repository of the Browser can be extended on demand. For this purpose we have to add to the GUI-Shell interfaces and functionality for context and MetaProvider descriptions exchange and editing.

As was mentioned before, we have at least two types of context. Talking about the second type contexts we have contexts with various different descriptions. In current version we are not talking about automatic context creation. Now GUI-Shell provides the interface for creation and editing of contexts for resource Closeness Context. These interface and correspondent logics are imbedded functionality of the Browser. But, if we consider gradual extension of local repository with context descriptions, we face a problem of functional extension of the Browser. We need somehow add/imbed the logic and functionality to make Browser able to manipulate with new contexts. At the same time with extension of local repository of context descriptions, we have to extend a programming code of the Browser (add some JavaScript code, create and connect Server part of the functionality).

To solve this problem, 4I Browser infrastructure can be constructed based on modular approach that is used for example by award-winning content management system[10] Joomla[11], which enables you to build Web sites and powerful online applications. Many aspects, including its ease-of-use and extensibility, have made Joomla the most popular Web site software available. A major advantage of using a CMS is that it requires almost no technical skill or knowledge to manage. Thus, powerful application framework of such a system makes it easy for developers to create sophisticated add-ons that extend the power of it into virtually unlimited directions. Following such approach, definition of new context will be accompanied not only by correspondent description, but also by JavaScript code, correspondent functional server part and appropriate html, media and other files. Then extension of the Browser with new context will be conducted via installation of certain package (Figure 5.3).

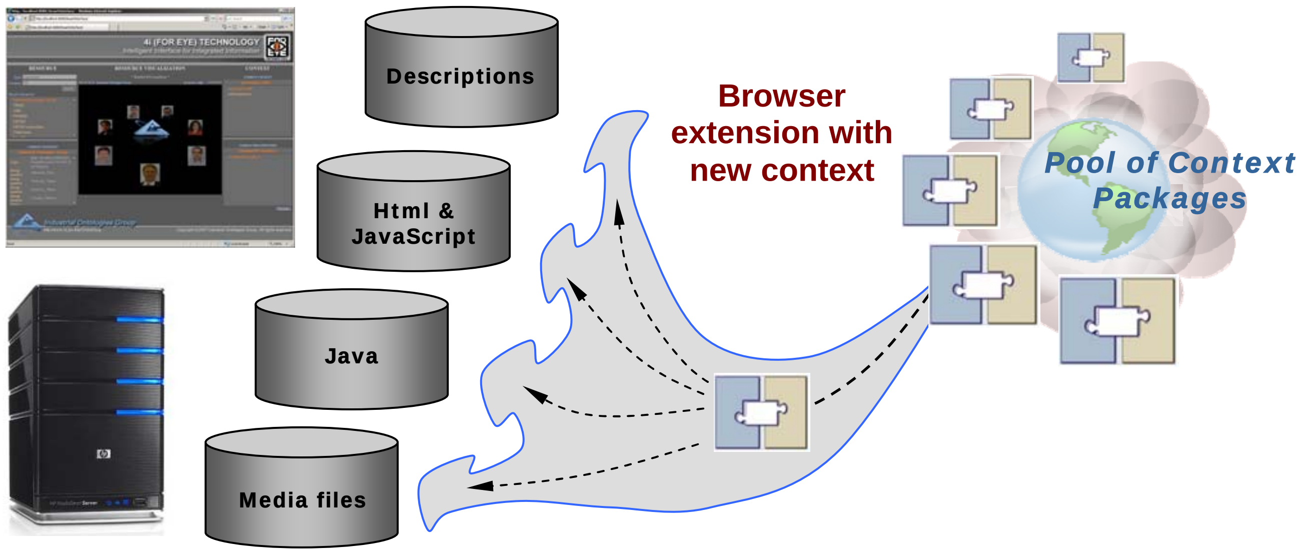

***Figure 5.3** – Browser extension with new context.*

Extension of the Browser with new MetaProvider is much simple (see Figure 5.4). We just have to extend the repository of MetaProvider's descriptions with a new one. MetaProvider description is standardized. It describes address, in-parameters and output (resource ID).

[10] Content Management System (CMS) - http://en.wikipedia.org/wiki/Content_management_system
[11] Joomla - http://www.joomla.org/about-joomla.html

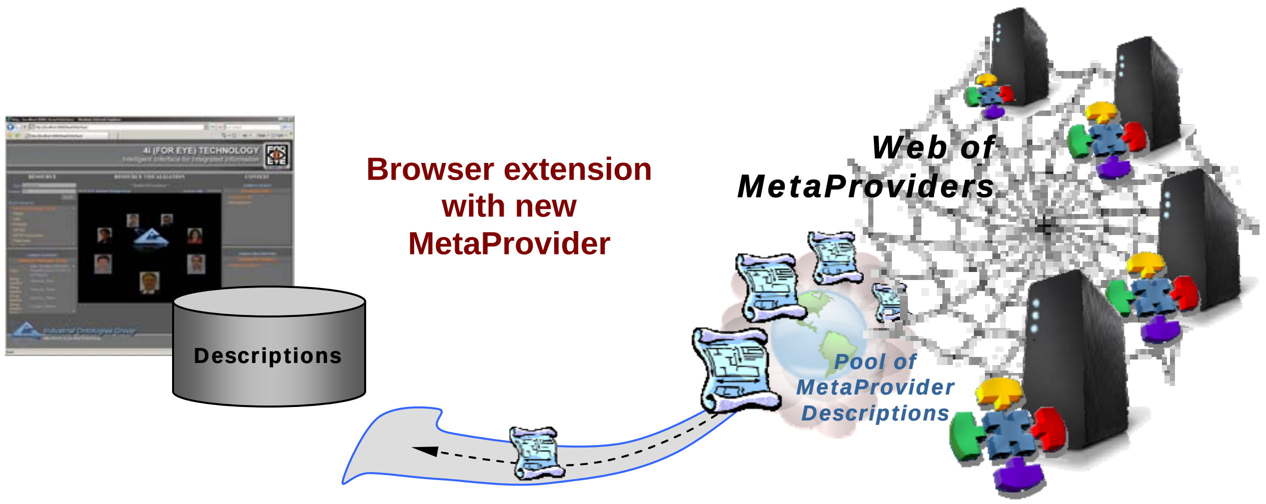

***Figure 5.4** – Browser extension with new MetaProvider.*

5.2 Commercialization ways of the Browser

5.2.1 General business model

In this section we are going to consider whole picture of 4I Environment from the business point of view. Usually business environment is divided to a number of arias of influence and roles that can be played by different parties. In 4I Environment we highlight such roles as (see Picture 5.5):

- **Browser Provider** Entity (organization) that provides GUI-Shell and all specifications of 4I Browser, supportive infrastructure and tools for 4I Browser components development.
- **Holder of the Browser** Entity that holds Browser (GUI-Shell): administrates, extends infrastructure with the access to new components (Resources, Contexts, MetaProviders). Holder provides user access to the Browser and performs personalized profiling of the users.
- **Provider of MetaProvider** This is owner of visualization module (MetaProvider). Such kind of service provides access and gives a result accordingly to the specification of 4I Browser that relates to MetaProvider development. Provider registers MetaProvider in appropriate MetaProvider Registry to be found and used by GUI-Shells. Depending on type of MetaProvider, it can have access to some resource information repositories relevant to MetaProvider specifics.
- **MetoProvider Registry Holder** Entity that provides registry place for MetaProviders and plays a role of mediator between separate MetaProviders and GUI-Shells to allow second one to find appropriate visualization module based on semantic description.

- **Context Generator & Provider** Institution that is responsible for context creation accordingly to the specification of 4I Browser and provides access for GUI-Shells to extend local repository with new contexts.
- **Resource Information Holder** This is a holder of information about resources in a form of semantic resource descriptions.
- **User** User that gets all the benefits provided by 4I Browser.
- **Ontology Holder** Entity that holds local (domain) or global ontology that plays a role of basis for correspondent infrastructure of 4I Environment and associated parts of the 4I Browser.

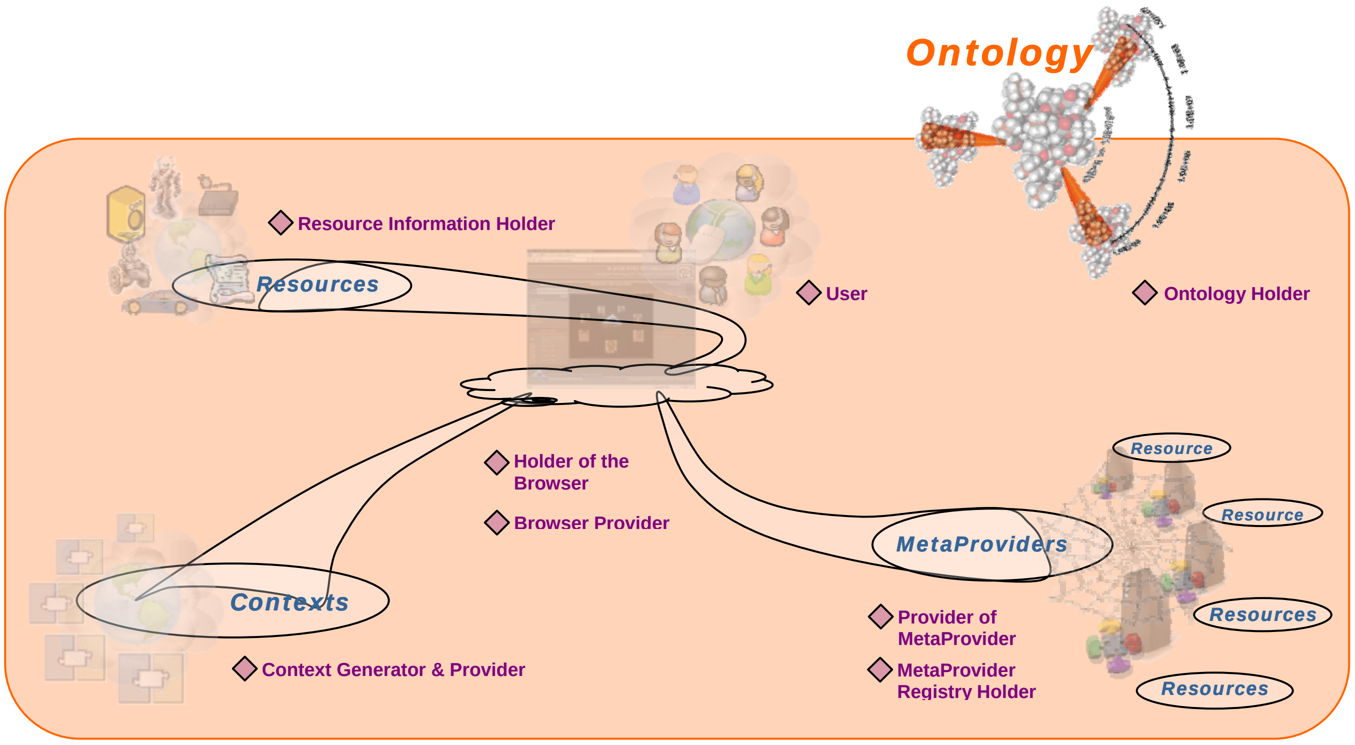

***Figure 5.5** – General model of 4I Environment.*

Of course, it does not mean that each role should be plaid by one separate player (party). Usually some of parties play several roles depending on business model and chosen strategy.

5.2.2 Business scenarios

In this section we describe some business scenarios that can be built within present infrastructure and show business potential of 4I Environment.

5.2.2.1 Scenario 1 – Global Use of the Browser

In this scenario Browser is used as a web-based service like an add-on to some Semantic Search System or as a separated service that uses access to the semantically annotated resources provided by such System. It could be a kind of new "Context-aware Visual Google".

Considering this scenario we have following set of players:

- **Player 1** is Semantically-enhanced Search System that plays role of *Resource Information Holder*.
- **Player 2** provides 4I Browser as a service. It plays role of *Holder of the Browser* as well as role of *Context Generator & Provider* and *MetaProvider Registry Holder*. The last two roles can be outsourced to another party (player), but the easiest way to avoid the problems with stable work of whole system is to hold these roles in one hands. In case of Global Use of the Browser all these roles should be played by *Browser Provider*.
- **Player 3** is a list of parties that play roles of *Providers of MetaProviders.* Thus, there are can be as many as possible players that play such role. This is an open environment and these players provide remote context-aware resource visualization service.

Considering the role of *Ontology Holder*, this is not a trivial task to entrust this role to some of the player(s). In this case ontology should be common and shared by all the players. The most probable case when the role of *Ontology Holder* is shared by *Player 1* and *Player 2* and *Player 3* just follows this ontology. The most promising case is when *Player1* and *Player 2* is one merged Player and autocratically takes care of ontology.

5.2.2.2 Scenario 2 – Local Corporate Use of the Browser

The main difference from the previously described scenario is that 4I Browser is used for domain specific resource browsing inside an organization. Such scenario is more suitable for the first step of commercialization and use, because the creation of correspondent domain specific ontology (as a one of the biggest challenges) can be performed in a short term depending on a task, and extended later on demand. From the other side current scenario entrusts almost all the roles and responsibilities on the organization. Even role of *Providers of MetaProviders* in most cases should be played by this organization due to domain specifics of the tasks. But it is still possible to outsource visualization part to another third party and use MetaProviders as external services. Thus, this scenario becomes a particular (much smaller) case of the most promising case of the previous scenario with merged players (*Player 1* and *Player 2*) and autocratic care of ontology.

Considering this scenario we have following set of players:

- **Player 1** is a *Browser Provider* that provides whole infrastructure as a full-fledged system in one package with all necessary specifications and supportive documentation.

- **Player 2** provides 4I Browser as a service. It plays role of *Holder of the Browser*, should provide access to a repository of semantically annotated resources as a *Resource Information Holder*, and play the roles of *Context Generator & Provider* and *MetaProvider Registry Holder* as well. As was mentioned before, due to domain specifics, it should play a role of *Providers of MetaProviders* also.
- **Player 3** is a list of parties that play roles of *Providers of MetaProviders.* Even taking into account that we are dealing with domain specific tasks, 4I Environment is an open environment and these parties can provide context-aware resource visualization outsourced by *Player 2*.

Current scenario can evolve to a Web of separated Corporate 4I Environments later. In this case those parties who play roles of external *Providers of MetaProviders* will be interested to provide own services more then one customer. At the same time, Players, who are holders of Corporate Environments, will be interested to utilize more advanced external visualization services that belong to another Corporate Environment. At that point we will face a problem of interoperability of the systems especially on the level of ontology specification, resource annotation and context creation. Thus, with further evolution of the scenario, some new Players can emerge: those who will take care about common ontology or bridging technology for already existing, those who will be responsible for common Registry of MetaProvider and common context creation and provisioning.

5.3 Enhancement of the Browser

In current Inno-W company case we are working on context aware search system that discover similar resources based on configurable semantic distance function and visualizes the result in interactive and handy for user way. Inno-W company holds corporate knowledge as a repository of structured documents, that present project ideas/proposals (those can play a role of resources in considered search system) and willing to utilize such a system to rank and find similar and relevant resources. This allows management group to find similar proposals and combine the groups with a purpose to direct common efforts to one direction, to find the close needs and propositions to allow customer and executor meet each other, to rank the documents for various purposes. But in each case such a search process should be done in a certain context, with a certain preferences. So, the aim is to elaborate search system that will discover similar documents based on semantic distance measuring function taking into account contextual information to rank similar documents and avoid irrelevant results.

According to the classical theory, the whole world can be divided into three parts. The first part contains substances that directly influence the result of a function and are directly relevant as input parameters. In other words, they are parameters that should be given to the function; otherwise it can not provide a result. The second part is formed of substances those influence on a function itself, configure the function and improve the result depending on the situation (context). And the third part is formed of irrelevant substances, which have no influence on the result of the function. Talking about the case above, contextual information play role of a filter and help to configure search function to present more relevant (in current context) results.

To improve knowledge acquisition, transfer, sharing and reuse we define contextual information and have to elaborate a model of influence of this information on resources representation, model that configures the resources representation function based on contextual information. There are several approaches that can be used to define a model of influence of the context: supervised and unsupervised machine learning algorithms, theory based constructions, etc.

As a next valuable enhancement of the Browser we consider intelligent way of automatic/semiautomatic context recognition and personalized visualization invocation. In current prototype we use just already predefined contexts that further are selected by user to visualize resource in appropriate way. When we are talking about automatic context generation, we should have a model that represents a context and can be used by machine learning algorithms to automate the process. Thus, to elaborate a mechanism of automatic/semiautomatic context generation, we are considering a common approach for the models of context where: context is a filter of resource representation in certain situation (state); resource is characterized by set of properties/attributes as well as other relevant to the resource resources in particular context. Formal model of the context in this case is a set of coefficients/weights that shows correspondence of the attributes, relevance of them in certain sense (Figure 5.6).

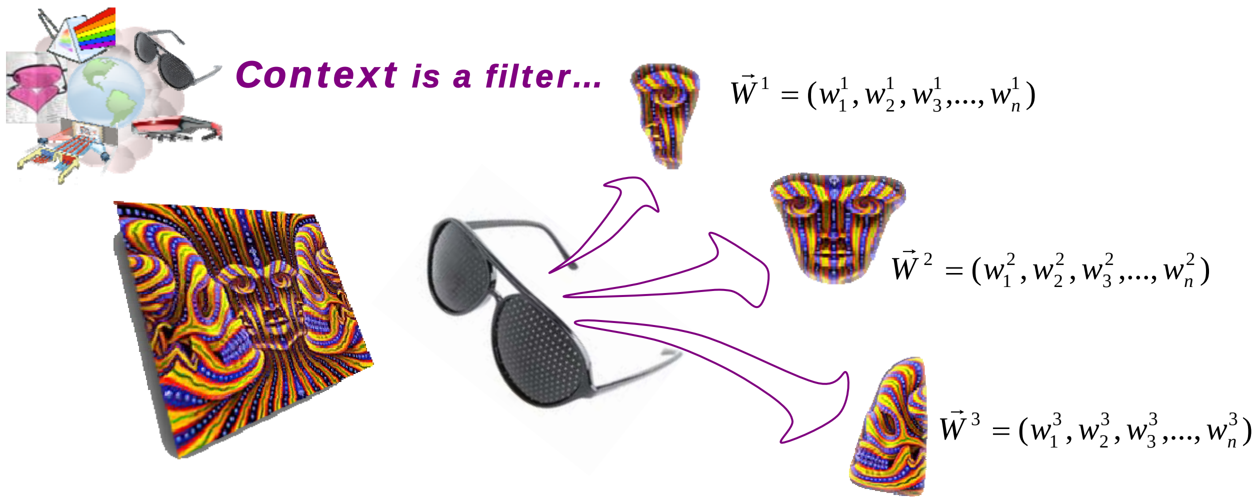
Context is a filter...
$\vec{W}^1 = (w_1^1, w_2^1, w_3^1, ..., w_n^1)$
$\vec{W}^2 = (w_1^2, w_2^2, w_3^2, ..., w_n^2)$
$\vec{W}^3 = (w_1^3, w_2^3, w_3^3, ..., w_n^3)$

***Figure 5.6** – Weight based context representation model.*

If we consider a context as a set of weights $Context = \vec{W} = (w_1, w_2, w_3, ..., w_n)$, then visualization modules should take these weights as input parameters and provide correspondent visual representation of a resource in particular context.

Figure 5.7 shows us the ways of information differentiation for learning samples creation and shows the whole picture of learning process of the system to be able to perform automatic/semiautomatic context recognition and build a context model later. It shows the history of user driven performance of the system. This is a good learning sample for machine learning algorithms to build a model for automated context recognition.

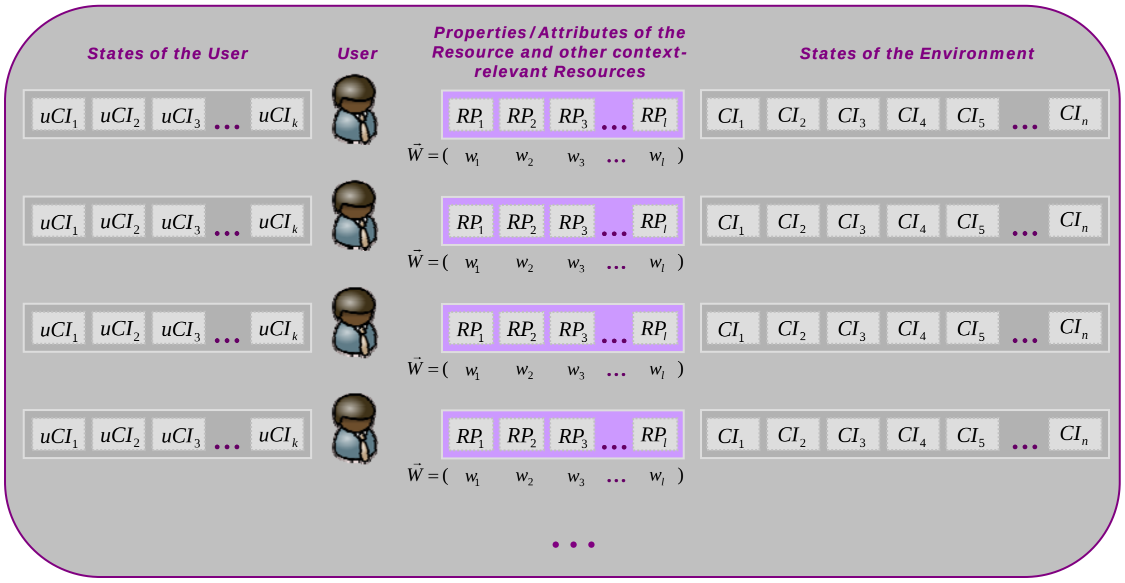
States of the User
User
Properties/Attributes of the Resource and other context-relevant Resources
States of the Environment
uCI_1 uCI_2 uCI_3 … uCI_k
RP_1 RP_2 RP_3 … RP_l
$\vec{W} = (w_1 \ w_2 \ w_3 \ \dots \ w_l)$
CI_1 CI_2 CI_3 CI_4 CI_5 … CI_n

***Figure 5.7** – Learning model for automatic/semiautomatic context recognition.*

Three sets of information needed for learning algorithms are presented on the figure:

- **State of the Environment** is a set of contextual information $(CI_1, CI_2, CI_3, CI_4, CI_5, ..., CI_n)$ (properties/attributes of resources that could be contextual);
- **State of the User** is a set of properties/attributes $(uCI_1, uCI_2, uCI_3, ..., uCI_k)$ of the user (his/her goal, location, different statuses, current responsibilities and duties, and etc.) that also could be contextual;
- **Properties/Attributes of the Resource and other context-relevant Resources** $(RP_1, RP_2, RP_3, ..., RP_l)$. They are data that could be under the influence of contextual information. Thus, the relevance level of Resource properties (context model) in current situation is a vector of weights $\vec{W} = (w_1, w_2, w_3, ..., w_l)$.

At the first stage, system should work in user-driven manual mode to collect a huge enough set of learning samples. System should collect all the Stages of Environment and User as well as vector of weights that User specifies based on own experience and knowledge. When we have collected enough data, there are several strategies that can be used to learn the system to automatically build context model:

- **Strategy 1:** As a simple case, we can consider *State of the User* as a part of *State of the Environment*. Then we will build the model of $(CI \cup uCI)$ influence on a vector of weights $\vec{W}$. Unfortunately such approach does not give us an ideal model, because a user affects an influence of environment on model (in other words, user is a contextual entity with indirect influence on the model).

- **Strategy 2:** Here we are taking into account an influence of the user on a model and consider *State of the User* as a separate set of contextual information (let us say – meta-level context (Khriyenko, 2008). Thus, at the first step we have to find correlation between *State of the User* and *State of the Environment* and then build two-level influence model.
- **Strategy 3:** Probably the most correct approach is a combination of previous two. We have to consider *State of the User* as a part of *State of the Environment* and at the same time use it as a meta-context to find the most relevant contextual information (data) for resource visualization depending on the User context.

One of the challenges that we can face elaborating such system functionality is how to collect *State of the User* and *State of the Environment*. Depending on domain and arias of system usage amount of the contextual resources, their properties can vary. But the algorithm can be elaborated more generally to fit any size of the data sets.

5.4 Conclusions and future work

To become a product, prototype should pass through a number of stages that bring generalization to the prototype (ability to be used in different domains and arias), provide common information infrastructure and user/programming interfaces of the system to allow extension and configuration of it. One of the challenges is ontology creation task. No doubts that the best way is to develop common ontology. But, practically, more realistic way is to start from building local domain and task-specific ontologies and further apply adaptation and bridging technologies to allow interoperability between systems based on different ontologies (Pinto and Martins, 2001), (Keet, 2004), (Alasoud et. al., 2005). Current prototype does not have separate ontology as such, and uses some limited (task specific) imbedded one. Thus, ontology/vocabulary creation and manipulation mechanism is one of the challenges that has to be taken into account.

As was mentioned before, ability of the Browser to connect/import external information repositories is very important functionality. GUI-Shell should provide user interface to select external repository to be imported via appropriate adaptation module from a list of available in Browser as well as interface for search of new adaptation modules and their imbedding. Regarding to the plan for Inno-W industrial case, we are going to elaborate adaptation module to convert their RDF repository into internal format. It will be just one adaptation module, but we are going to implement general functionality of the Browser according to modular approach of source data adaptation.

The same modular approach also can by apply for resource visualization context definition and creation. Following such approach, definition of new context will be accompanied not only by correspondent description, but also by JavaScript code, correspondent functional server part and appropriate html, media and other files. Then extension of the Browser with new context will be conducted via installation of add-on package. Extension of the Browser with new MetaProvider could be much simple and be conducted via extension of repository of MetaProvider's descriptions with a new one.

Concerning the commercialization steps of the Browser, we have considered general model of 4I Environment where we tried to define main players and roles. We described two business scenarios of Browser utilization: *Global Use of the Browser* and *Local Corporate Use of the Browser*. So, as in a case of ontology creation we think that the easiest way to start with the second scenario, but this scenario can evolve to a Web of separated Corporate 4I Environments later. In this case we will face a problem of interoperability of the systems especially on the level of ontology specification, resource annotation and context creation. Thus, with further evolution of the scenario we will come to the first scenario with more complex structure. Still, it is more realistic scenario, but of course, the best scenario is the first one, even if it demands more efforts and investments.

Finally, we have presented intelligent way of automatic/semiautomatic context recognition and personalized visualization invocation as a next valuable enhancement of the Browser. This is a first draft of idea and should be elaborated more comprehensive in the future.

UBIWARE Deliverable D3.1:
Workpackage WP6:

6 Middleware for Peer-to-Peer Discovery (*postponed*)

The objective of this workpackage is the design of mechanisms which will extend the scale of semantic resource discovery in UBIWARE with peer-to-peer discovery. Such mechanisms have to enable an agent:

- To discover agents playing a certain organizational role,
- To discover an agent (or agents) possessing certain needed information.
- To discover resources (through its agents) of certain type or possessing certain properties (e.g. a device in state X, or a web-resource providing some information searched for).

In all cases, existence of a central Directory Facilitator is not assumed, so the discovery is to be performed in peer-to-peer manner. This has at least two goals:

- Improving the survivability of UBIWARE-based systems. P2P is a complementary mechanism which can be utilized in an exception situation where the central Directory Facilitator became for some reason unavailable.
- Discovery across UBIWARE-based systems. There could be several UBIWARE-based agent platforms (applications) started up independently, each having own Directory Facilitator. While JADE (underlying agent framework of UBIWARE) supports communication among agents residing on different platforms, it does not provide for the cross-platform discovery.

Because of reduced funding, it is considered to be reasonable not to perform work in this WP during Year 3.

REFERENCES

Alasoud, A., Haarslev, V. and Shiri, N. (2005) A Hybrid Approach for Ontology Integration, Proceedings of the 2005 VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS-2005), Trondheim, Norway, Sept. 2, 2005, pp. 18-23. URL: http://users.encs.concordia.ca/~haarslev/publications/ODBIS-05.pdf

Bleier, A., Ivanchenko, Y., Katasonov, A., Kaykova, O., Kesäniemi, J., Khriyenko, O., Nagy, M., Nikitin, S., Szydlowski, M., Terziyan, V. and Vapa M. (2008) Individual resources and inter-resource communication in UBIWARE, Technical Report (Deliverable D 2.1), UBIWARE Tekes Project, Agora Center, University of Jyvaskyla, May-October 2008, 116 pp.

Dey, A. K. (2001) Understanding and Using Context, Personal Ubiquitous Comput., Springer-Verlag, Vol. 5, pp. 4-7

Dobson, S., Denazis, S., Fernández, A., Gati, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F. (2006) A survey of autonomic communications, ACM Trans. Auton. Adapt. Syst., ACM, Vol. 1, pp. 223-259

Garlan, D. and Schmerl, B. (2002) Model-based adaptation for self-healing systems, In Proceedings of WOSS '02: First workshop on Self-healing systems, ACM, pp. 27-32

Gat, E. (1997) On Three-Layer Architectures, Artificial Intelligence and Mobile Robots

Guazzelli, A., Zeller, M., Lin, W. and Williams, G. (2009) PMML: An Open Standard for Sharing Models. The R Journal, Volume 1/1, May 2009.

Keet, C.M. (2004) Aspects of Ontology Integration, Literature research & background information for the PhD proposal entitled: Bottom-up development of ontologies and ontology integration in the subject domain of ecology, Scotland, 2004. URL: http://www.meteck.org/AspectsOntologyIntegration.pdf

Kephart, J. O. and Chess, D. M. (2003) The Vision of Autonomic Computing, Computer, IEEE Computer Society Press, Vol. 36, pp. 41-50

Khriyenko, O. (2007a) 4I (FOR EYE) Technology: Intelligent Interface for Integrated Information. In Proc. of the 9th International Conference on Enterprise Information Systems (ICEIS-2007). Funchal, Madeira – Portugal, 2007.

Khriyenko, O. (2007b) Context-sensitive Multidimensional Resource Visualization. In Proc. of the 7th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2007). Palma de Mallorca, Spain, 2007.

Khriyenko, O. (2008) Adaptive Semantic Web based Environment for Web Resources, PhD.

Thesis, Department of Mathematical Information Technology, University of Jyväskylä, Vol. 97, December 2008.

Kramer, J. and Magee, J. (2007) Self-Managed Systems: an Architectural Challenge, In Proceedings of FOSE '07: 2007 Future of Software Engineering, IEEE Computer Society, pp. 259-268

Pinto, H.S. and Martins, J.P. (2001) A Methodology for Ontology Integration, In Proc. of the 1st international conference on Knowledge capture, Victoria, British Columbia, Canada, 2001, ISBN:1-58113-380-4, 131-138 pp.

Quitadamo, R. and Zambonelli, F. (2008) Autonomic communication services: a new challenge for software agents, Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, Vol. 17, pp. 457-475

Shaw, M. (1995) Beyond objects: a software design paradigm based on process control, SIGSOFT Softw. Eng. Notes, ACM, Vol. 20, pp. 27-38

Witten, I. H. and Frank, E. (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

Appendix A: UBIWARE Publications List

(up to the October of year 2009)[*]

[1] 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, ISBN: 978-0-88986-800-7, pp. 184-191.

[2] 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).

[3] 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).

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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).

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.