UBIWARE Deliverable D2.1:

Individual resources and inter- resource communication in UBIWARE

October, 2008

Date	October 29, 2008
Document type	Report
Dissemination Level	UBIWARE project consortium
Contact Author	Vagan Terziyan
Co-Authors	Arnim Bleier, Yevgeniy Ivanchenko, Artem Katasonov, Olena Kaykova, Joonas Kesäniemi, Oleksiy Khriyenko, Michal Nagy, Sergiy Nikitin, Michal Szydlowski, Mikko Vapa
Work component	WP1, WP2, WP4, WP5, WP6
Deliverable Code	D2.1
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 D2.1 reports on the research results from work packages WP1, WP2, WP4, WP5 and WP6 (it was decided not to perform the work at the WP3 during the second project year due to reduced resources).

The further text describes new developments on scientific concepts and results of the second 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 B).

UBIWARE Deliverable D2.1:
Workpackage WP1:
Task T2.1_w1:
Workpackage leader: *Artem Katasonov*

1 UbiCore – Core Distributed AI platform design

The main objective of the core platform is to ensure a predictable and systematic operation of the components and the system as a whole by

- enforcing that the smart resources, while might to have own "personal" goals, act as prescribed by the roles they play in a organization and by general organizational policies,
- maintaining the "global" ontological understanding among the resources, meaning that a resource A can understand all of (1) the properties and the state of a resource B, (2) the potential and actual behaviors of B, and (3) the business processes in which A and B, and maybe other resources, are jointly involved

In the core platform that resulted from the research during Year 1, the behavior of an agent is prescribed by a set of behavioral models written in Semantic Agent Programming Language (S-APL). So far, it remained the full responsibility of the designer to ensure the absence of conflicts between different behavior models, e.g. roles, of one agent, as well as conflicts between different agents. During WP1's Year 2 (the *Deliberation* phase), the goal is to put more conflict-resolution intelligence into the platform, so that the agent would be able to resolve conflicts dynamically rather than rely on the absence of conflicts. The original plan for Year 2 included the following research questions:

- How to realize organizational policies as restrictions put on the behavior of individual agents?
- What mechanisms are needed for flexibly treating the potential (and likely) conflicts among the behavioral models (roles, policies) used by one agent?
- Is it possible and if yes then how to implements a mechanism so that agents could "see" what other agents are doing, in so creating the basis for coordination through observation in addition to traditional coordination through communication?

The work related to the first question has resulted in that the scope of the considered problem has been extended towards more general ontological coordination among autonomous agents. The traditional policies, such as of access control, are considered as a special case of coordination: an agent with an authority sets some restrictions on the behavior of other agents to avoid possible conflicts of interests. A more general case of coordination is where only when two autonomous behaviors conflict over the use of a resource some policy mechanism is applied. Such a resolution policy may dictate, for example, that the agent with a higher organizational authority gets the priority in using the resource.

The work related to the second question also resulted in somewhat extending the scope of the problem considered and shifting the focus – towards general-case agent's decision making when confronted with several options. Selecting among actions dictated by different roles is a special case of this.

The sections 1.1, 1.2, and 1.3 describe the research results achieved with respect to the there questions above, correspondingly.

1.1 Ontological Coordination in Multi-Agent Systems Built with S-APL

The research results of this Section are presented in the form of a separate scientific article. To avoid reformatting, the article is included as APPENDIX A of this report.

1.2 Market-based Evaluation of Alternative Actions

In this section, we try to answer the following detailed questions: (1) How to specify behavior in terms of high-level objectives, at the design time, but avoiding over determination at the run-time? (2) How to evaluate the performance of an agent in regard to its behavioral models? (3) Is a mapping possible between the behavioral models of a group of agents and its members and if yes how to formalize it? (4) What mechanisms are needed to resolve potential (and likely) conflicts among behavioral models of an agent or a group of agents?

1.2.1 Theoretic Foundations

In this first section, the theoretical foundations are lined out. Thus, in the first paragraph Utility Functions are introduced (question 1). In the next paragraph, the deployed utility model is shortly investigated (questions 2 and 3). In the last paragraph a mechanism is introduced to foster joint acting (question 4).

1.2.1.1 Utility Functions

Utility Maximizing Agents are widely recognized as the most developed type of software agent, and Utility Function strategies codify their agenthood. Utility Function strategies determine a real-valued scalar for a given state of the agent's knowledge base (KB), expressing the preference for facts in the KB of an agent. Those strategies enable the exact specification of tradeoffs; thus, the decision-making in potential conflict situations. However, the challenge in the design of Utility Function policies is to specify an n-dimensional set of facts on which preferences are imposed as well as to find an optimal solution for the Utility Function.

1.2.1.2 A Utility Model

Subsequently it will be referred to a set of statements that describe a potential state of the KB as context C of the attributes A. Thus, a set of attributes $A = \{A_1, A_2, ..., A_n\}$ is a subset of a context C. The values a_j of attribute A_j can either be discrete $a_j \in \{a_{j1}, a_{j2}, ..., a_{jm}\}$ or continuous $a_j \in \{\min_j \leq a_j \leq \max_j\}$. The possible attribute space S is an n-dimensional set characterized by the Cartesian product $S = A_1 \times A_2 \times ...A_n$. Consequently, Utility Functions that map the attribute space, within a context, to a real-valued scalar are multiple parameter functions. A concrete set of attributes is denoted by $s \in S$.

A preference structure is constructed by a transitive, reflexive and complete binary relation ≥. The structure of preferences can be inferred from an evaluation function $f : S \to R$, that maps a state to a real-valued scalar, where the condition $\forall s_x, s_y \in S : s_x \geq s_y \equiv f(s_x) \geq f(s_y)$ holds.

However, the number of possible attribute combinations grows exponentially with respect to the attributes and their values. Fortunately, in many practical situations the attributes are mutual independent, also known as additive severability. The assumption of additive severability does hold if and only if there exist functions $f_1, f_2, ..., f_n$ such that $f(a_1, a_2, ...a_n) = f_1(a_1) + f_2(a_2) + ... + f_n(a_n)$. Relying on additive severability improves the compactness and manipulability by decomposing $f(S)$ into multiple one-dimensional functions.

In economics two different notions of the utility concept are distinguished; the ordinal utility concept and the cardinal utility concept. When an ordinal utility notion is used differences in the valuations carry only the information of the preference ordering between the members of a choice set (i.e. the different sets of attributes whose utilities should be compared). Thus, ordinal utility theory can only be used to determine which of the alternatives is the best one. The cardinal utility concept carries the objective utility of a concrete set of attributes; thus, allows deciding whether a set of attributes is describing something whose utility is big enough. In particular, the cardinal utility concept allows the comparison of levels of satisfaction across different agents; if a particular item gives one agent 4 utility units but another gets 2 from the same thing, the item described by a set of attributes is said to give the one twice as much as the other one. This sort of comparison is of great use in Multi-Agent Systems when it comes to the evaluation of how well the system is doing overall. Consequently, under the framework of utilitarianism actions are judged by their contribution to the reaching of the overall goal of the system.

1.2.1.3 Auctions

In the previous paragraph, a mechanism that is potentially capable of determining the value of actions in terms of utility, associated with the state reached upon the application of an action, has been introduced. However, a negotiation mechanism is needed, if an action has not only positive consequences but also negative ones, to determine weather an action leads to an overall increase of utility and who has to pay the costs. A negotiation mechanism can be seen as such a protocol, prescribing how agents interact to determine a contract. This work uses the mechanism of the Second Price auction, also known as Vickrey auction, in which bidding agents submit their bids without knowing the bids of the other agents. The highest bidding agent wins, but the price to be paid is the one in the second highest bid. The idea of this mechanism is that each agent in the auction pays the opportunity costs that their presence puts on all the other agents. Payments between agents are done by the use of Util's. Whereby, the valuation function $f(util) = util$ neglects the diminishing marginal utility of an additional Util for the sake of simplicity and to use the valuation derived by a Utility Function as the willingness to pay.

1.2.2 Modeling Atomic Utility Functions

In this section and the following ones, the incorporation of the research finding is in the centre; it is started with the functions for the evaluation of attributes. To acknowledge the different types of attributes (i.e. discrete and continuous ones) not only one function is introduced, but three. In the first paragraph Discrete functions are introduced. In the next paragraph, Linear functions are introduced. In the last paragraph Polynomial functions are introduced. To give the reader a broad overview on the design of the functions the pervious figure (Figure 1.1) shows their ontology in graph form.

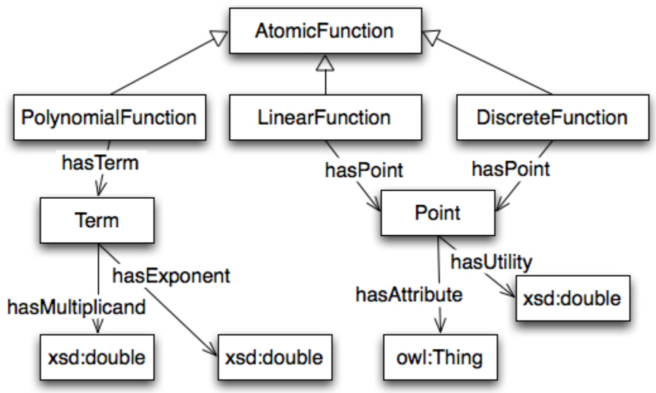


Figure 1.1

1.2.2.1 Discrete Functions

Functions of the `rdf:type org:DiscreteFunction` are connected to a set of individuals of the `rdf:type org:Point`, by the predicate `org:hasPoint`. Each individual point is connected via the property *`org:hasAttribute`* to a *org:Attribute,* and via the datatype property *`org:hasUtility`* to one utility value of the type *`xsd:double`*. Subclasses of `org:Attribute` can be used to define which values a specific attribute can adopt. Axiom A.1 and A.2 formalize this notion.

$$org:Point \subseteq owl:Thing \wedge_{=1} org:hasAttribute.owl:Thing \wedge_{=1} org:hasUtility.xsd:double \quad (A.1)$$

$$org:DiscreteFunction \subseteq org:AtomicFunction \wedge_{>0} org:hasPoint.org:Point \quad (A.2)$$

In the next step rules are required to evaluate the introduced concepts and their semantics. The formula below checks the set of points, of a discrete function f, to find the utility value for a given attribute value; thus, it declares how the value of a attribute can be determined.

```
{ sapl:I :calculate {?a ?f *}.
  ?f  rdf:type org:DiscreteFunction; org:hasPoint ?p.
  ?p org:hasAttribute ?ar; org:hasUtility ?u.
  ?a = ?ar.
} => {      sapl:I :calculate {?a ?f ?u}}
```

In the formula above the testing whether a given fact satisfies the attribute defined by a point is done by a simple equals-relationship. However, different types of attributes require different rules of entailment. Thus, this predicate should be altered according to the type attribute that is evaluated. While in some cases simple string matching is sufficient, other attributes demand a test whether at least the *rdfs:subclass*-relationship for the type of the attribute holds.

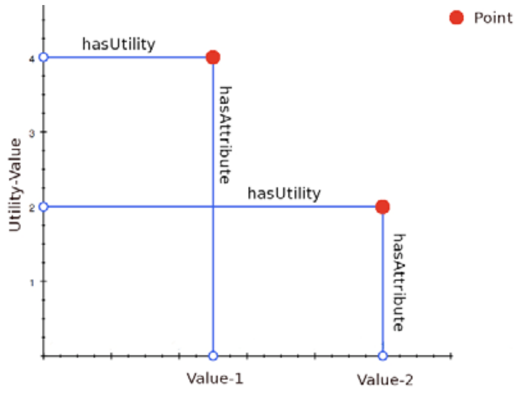


Figure 1.2

Ontologically modelling Functions is a relation between an `org:Attribute`, an instance of the class `org:AtomicFunction`, and a valuation to which the later maps the former. To represent such a three value relation the container in the statement `sapl:I :calculate _:containerID` has in its subject part the attribute, in its predicate part the Function URI, and in its object part the valuation of the subject by the predicate.

1.2.2.2 Linear Functions

Linear functions allow an ordering between given attribute values, of the type `xsd:double`, to specify a continuous range. They are an extension of the already introduced discrete functions in that way that points with neighbouring value attributes are connected by a line to make up a linear function. For each pair of points (x_1,v_1) and (x_2,v_2) as well as a given attribute value x, a utility value is calculated by the equation: $u = [(v_2 - v_1) * (x - x_1)]/(x_2 - x_1)$, for $x_1 < x \leq x_2$. Axiom A.3 formalizes the ontological status of the class `org:LinearFunction`.

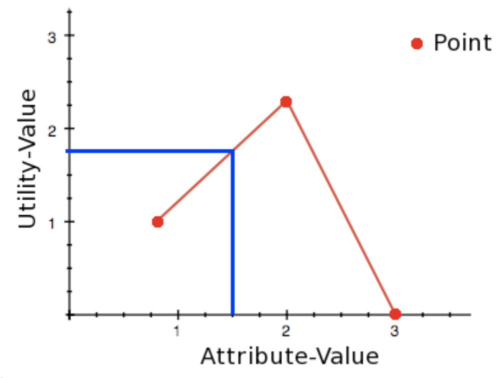


Figure 1.3

$$org:LinearFunction \subseteq org:AtomicFunction \wedge_{>1} org:hasPoint.org:Point \quad (A.3)$$

The formula, given above, calculates the utility u of a given attribute value v and ensures that only neighbouring points are taken into account for the calculation. The adding of the result of the function, to the KB, is analogue to the last paragraph.

```
{ sapl:I :calculate {?a ?f *}.
  ?f rdf:type org:LinearFunction; org:hasPoint ?1p, ?2p.
  ?1p org:hasAttribute ?1v; org:hasUtility ?1u.
  ?2p org:hasAttribute ?2v; org:hasUtility ?2u.
  ?1v < ?a. ?2v > ?a. ?v1v sapl:max ?1v. ?v2v sapl:min ?2v.
  ?1v = ?v1v. ?2v = ?v2v.
  ?u sapl:expression "?1u+(?a-?1v)*((?2u-?1u)/(?2v-?1v))".
} => {sapl:I :calculate {?a ?f ?u}}
```

1.2.2.3 Polynomial Functions

In addition to the previous paragraphs functions for continuous facts, such as the float datatype, can be modelled by means of polynomial functions. The class `org:PolynomialFunction` denotes functions constructed from one or more variables and constants using the operations of addition, subtraction, multiplication, and raising to constant non-negative integer powers. In general it can be written $f(x) = a_1 x^{p_1} + a_2 x^{p_2} + \ldots + a_n x^{p_n}$, where a_n and p_n represent parameters that have to be given to create the function.

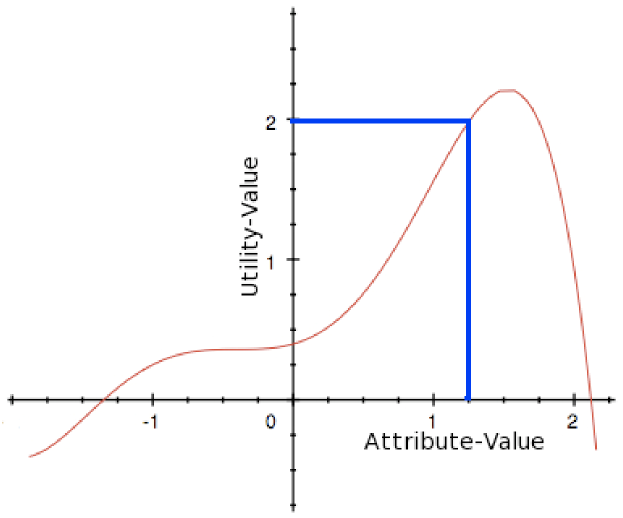


Figure 1.4

The ontology of polynomial functions is defined as follows. Instances of these function are connected to a set of terms by the predicate `org:hasTerm`; each individual term has exactly one multiplicand and one exponent. Axiom A.4 in conjunction with A.5 formalizes this notion.

$$org:\mathrm{PolynomialFunction} \subseteq \mathrm{org:AtomicFunction} \wedge_{>1} org:hasTerm.org:Term \qquad \text{(A.4)}$$

$$org:Term \subseteq owl:Thing \wedge_{=1} org:hasMultiplicand.xsd:double$$
$$\wedge_{=1} org:hasExponent.xsd:double \qquad \text{(A.5)}$$

The following formula calculates the sum of all terms specified for a polynomial function as well as the value of the terms itself. The adding of the result of the function is done likewise to the previous formulas.

```
{{sapl:I :calculate {?a ?f *}.
  ?f rdf:type org:PolynomialFunction; org:hasTerm ?p.
  ?p org:hasMultiplicand ?w; org:hasExponent ?e.
  ?x sapl:count ?p } sapl:All ?p.
} => {{      ?u sapl:expression "?w*pow(?a,?e)"
  }=>{ ?p :hasResult ?u}.
          {* :hasResult ?zu. ?y sapl:count ?zu.
          ?y = ?x. ?h sapl:sum ?zu.
  }=>{ sapl:I :calculate {?a ?f ?h}}}.
```

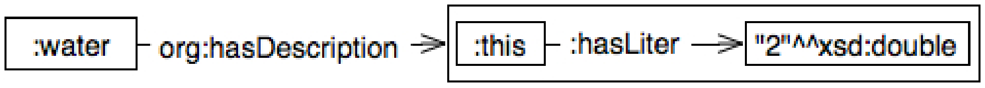
1.2.3 Mapping a Graph to a Value

Preferences of an agent towards states of its KB can be expressed upon Utility Functions that map a set of RDF statements, describing the current state of its KB, to a real-valued scalar. These Utility Functions provide a view on the world, from the perspective of the agent in the way that they reduce the complexity of the state of the world, of which the KB of the agent is a model of, to a one-dimensional utility valuation. Such utility valuations can be used to compare two states of the world. Thus, the comparison of utility valuations, for states of the KB before and after a hypothetical action, indicate the value of an action for the agent.

1.2.3.1 A Makeup for States of the World

Since it can be assumed that an agent has only a preference ordering for those parts of its KB that describe states of the world, a mechanism has to be introduced that distinguishes the states of the world from the rest of the KB. Individual states are made up by the class `org:Thing`. Each state has exactly one container associated, describing what the state is, as formalized in by axiom A.6. For example consider water owned by an agent.

$$org:Thing \subseteq owl:Thing \wedge_{=1} org:description.sapl:Container \qquad \text{(A.6)}$$



1.2.3.2 Modeling Complex Functions

In the next step the definition of multi attribute functions itself is in the focus. Multi attribute functions are modelled not as instances of a class, but as subclasses of the class `org:ComplexFunction`. This is because such a function is used to be matched not only against one single state, but to specify a class of states to whom it can by applied to; i.e. it specifies which individual states are instances that can be evaluated. The specification, which states satisfy the condition of evaluation, is done by a container connected to the class `org:ComplexFunction` upon the predicate `sapls:Restriction`. The container hosts those statements that have to be part of the state description to be evaluated by the complex function. Consider the graph in the upper part of the next figure (Figure 1.5) as an example. The description connected with the state in the previous figure satisfies the restriction, since all required statements, in the schema, are part of the description.

$$org:ComplexFunction \subseteq owl:Thing \wedge_{=1} sapls:\mathrm{Re}striction.sapl:Container$$
$$\wedge_{=1} org:has\mathrm{Associated}Functions.sapl:Container \qquad \text{(A.7)}$$

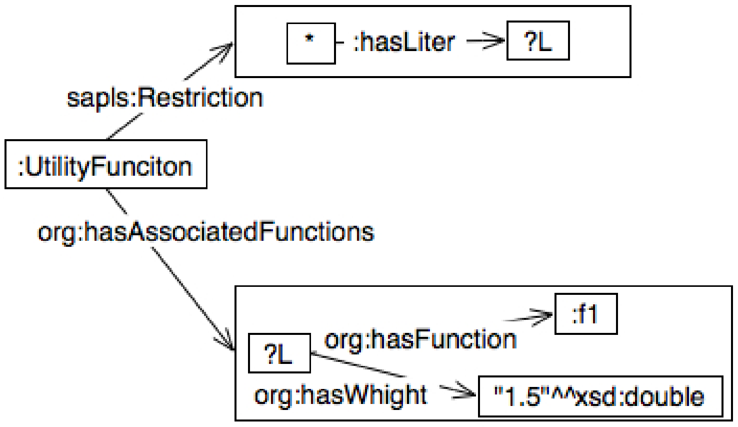


Figure 1.5

However, such a definition of a schema allows only to search for those states for which the Complex Function applies, but does not contain any information about how to valuate the statements in the description of a state. Thus, a further container is connected, to `org:ComplexFunction`, by the predicate `org:hasAssociatedFunctions` (axiom 7). This container has three purposes; first, those parts of the description are identified that are attributes; second, the identified attributes are connected to a real-value number upon the predicate org:hasWeight to determine their relative weight; third, the identified attributes are connected to an instance of `org:AtomicFunction` that is used for their valuation.

1.2.3.3 Evaluating Complex Functions

The first part of the evaluation formula is a rule triggered by a `org:ComplexFunction` in connection with its schema and using this schema then as part of its own antecedent. However, the traditional satisfy relation between a set of statements and an antecedent of a rule is no longer sufficient, but only a necessary condition, additional information for the calculation of the valuation of a graph has to be taken into account. Thus, in the next step the variables in the container hosting the functions, of a Complex Function, are connected to the variables in its schema part, by string matching of its names. This allows to derive for each variable (attribute) its associated weight and function. The body of the evaluation rule triggers then the evaluation of all attributes according to the appropriate org:AtomicFunction. The value calculated by the function, multiplied with the weight of the attribute, can be interpreted as the valuation that a single attribute contributes to the overall valuation of a Complex Function. In the last step of the sum of weighted individual valuations, of attributes, is calculated. The formula below shows the central aspects, as discussed.

```
{{sapl:I :calculate {?state ?complexFunction *}.
  ?complexFunction rdfs:subClassOf org:ComplexFunction;
            sapls:restriction ?schema;
            org:hasAssociatedFunctions ?functionContainer.
  ?state ?perdic {?schema sapl:is sapl:true }.
  ?functionContainer sapl:hasMember {
  ?attribute org:hasFunction  ?atribut_function;
            org:hasWhight ?atribut_weight }.
  {?schema sapl:hasMember {?match * *}. ?attribute = ?match} sapl:or {?schema
sapl:hasMember {* * ?match}. ?attribute = ?match}.
?NumberOfAttributes_1 sapl:count ?attribute } sapl:All ?attribute.
} => {
  sapl:I :calculate {?attribute ?atribut_function *}.
  {         sapl:I :calculate {?attribute ?atribut_function ?valuation}.
  ?weightValuation sapl:expression "?atribut_weight*?valuation".
  }=>{      ?attribute :hasValue ?weightValuation}.
  {         * :hasValue ?weightValuation.
```

```
            ?NumberOfAttributes_2 sapl:count ?weightValuation.
            ?NumberOfAttributes_2 = ?NumberOfAttributes_1.
            ?result sapl:sum ?weightValuation.
    }=>{sapl:I :calculate {?state ?complexFunction ?result}.
    ?state org:hasValuation ?result. }}.
```

The aggregation of the values for each attribute to the overall result and the adding of the result are done in the THEN-part. Analogue to the previous section modelling Complex Functions is a relation between a concrete description of a `org:Thing`, a `org:ComplexFunction` and a real-valued scalar, to which the later maps the former. In addition to the adding of the result into the container `sapl:I :calculate {}`, the thing evaluated upon a Complex Function gets connected to the result of the evaluation by the predicate `org:hasValuation`. The benefit of such a straight connection becomes visible in the next section.

1.2.4 Contracts

It is shown in this section how a utility valuation can be used to determine the value of actions, and to decide which actions should be made and which not. Since the context of this report is an economic one, the actions that will be considered are selling and buying. To establish a mutual beneficial agreement, in a potential cooperative situation between a seller and a buyer, a contract is needed to mark the terms to which both agree. Auctions are such a coordination mechanism to establish mutual beneficial agreements.

1.2.4.1 Auctions

In the first step it has to be determined how to offer something; thus, how to set up an auction. Since agents have to have the ability to determine the value of what the auction is about, the class `org:Auction` is modeled as a subclass of the class `org:Thing`. A Vickery auction, as a special kind of auction, is modeled as a subclass of the class auction as formalized in axiom A.8. and A.9.

$$org:Auction \subseteq org:Thing \wedge_{=1} \mathrm{org:hasAuctioneer}.sapl:Agent \qquad \text{(A.8)}$$

$$org:Vic\mathrm{ker}yAuction \subseteq org:Auction \wedge_{<2} org:hasDurnation.xsd:\mathrm{int}$$
$$\wedge org:hasMin\mathrm{Pr}ice.xsd:double \wedge org:hasBid.org:Bid \qquad \text{(A.9)}$$
$$\wedge_{<2} org:hasWinner.sapl:Agent \wedge_{<2} org:hasMarket\mathrm{Pr}ice.xsd:double$$

$$org:Bid \subseteq_{=1} org:hasAgent.sapl:Agent \wedge_{=1} org:hasValuation.xsd:double \qquad \text{(A.10)}$$

$$org:hasMin\mathrm{Pr}ice \subseteq org:hasValuation.xsd:double \qquad \text{(A.11)}$$

The predicate `org:hasAuctioneer` connects an auction with the name of the agent making the offer. The parameter `org:hasDuration` acknowledges the pragmatic assumption of not having a point market, but giving the agents time to response. The parameter `org:hasMinPrice` is the restriction that the Utility Function, of the auctioneer, puts on the potential action of selling something; in other words, a policy constraining the acceptance of offers. Thus, the datatype property `org:hasMinPrice` is a sub predicate of `org:hasValuation` as discussed in the previous section, see Axiom A.11.

1.2.4.2 Bidding

An agent can determine if it has received an offer by searching for instances of the Vickrey auctions whose property `org:hasAuctioneer` is different from its own name. If an agent finds such an offer a second evaluation of the auction starts. However this time, not by the Utility Function of the seller, but upon its own Utility Function a valuation is calculated. Thus, in this stage an agent is in the position to compare its own utility valuation, for the thing the auction is about, with the minimal price set by the seller. Likewise to the seller's side the Utility Function determines now in which potential trades (actions) the agent is interested in, as well as the policy for the maximal willingness to pay. In the next step an agent makes a bid by creating an instance of the class `org:Bid` and connecting it together with its name and its utility valuation to the auction.

1.2.4.3 The Evaluation

In the last step of an auctioning process the winner and the price to be paid have to be determined. If bids are made the next formula determines in the first step, which is the top bid; in the second step, which is the second highest bid; and in the last step who has made the top bid. In the consequent graph the property `org:hasWinner` and the datatype property `org:hasMarketPrice` are added to the auction; the first connecting the name in the top bid, the second connecting the valuation in the second highest bid.

```
{{      ?auct rdf:type org:VickreyAuction;
        org:hasMinPrice ?minprice;
        org:hasBid [org:hasAgent ?agent;
                org:hasValuation ?bidPrice, ?SecBidPrice].
        ?SecBidPrice < ?bidPrice.
        ?SecMaxBidPrice sapl:max ?SecBidPrice.
  ?auct org:hasBid [org:hasAgent ?winner;
  org:hasUtility ?SecMaxBidPrice].
}=>{?auct org:hasWinner [org:hasAgent ?winner;
         org:hasValuation ?SecMaxBidPrice]}.
```

The figure (Figure 1.6) shows an exemplary situation, utilizing the concepts introduced, so far. Agent Mary launches the auction of `:water` with the minimal pricing of 5 Utils. Agent John receives the offer, evaluates it, with its Utility Function upon the value of 10 Utils, and places his bid accordingly. The same applies to agent Bob; however, his Utility Function results only with a valuation in the height of 6 Utils. After both agents have made their bids, the auctioneer Mary determines the winner John and the price to be paid in the amount of 6 Utils, leading to an overall increase in the utility for the agents in the amount of 1+4=5 Utils.

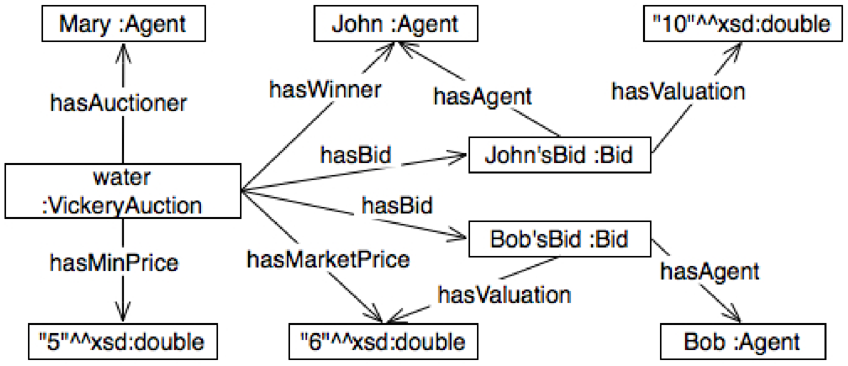


Figure 1.6

1.2.5 Dynamic Allocation

The approach discussed so far enables agents to express a utility for goods and to acquire them. However, the approach runs into a problem if not only one item can be acquired, but multiple ones. In other words, it is possible to express the utility an agent gets from the ownership of a print service, but it remains open how to model that an agent might have a lower benefit from acquiring an additional unit of the resource than in the first place.

To solve this the already introduced concept of auctions will be changed in two ways: First, the minimal bidding price and the bidding price itself will not longer be derived upon the valuation by the Utility Function, but by the difference a loss or gain (i.e. Border-use), of the thing the auction is about, would bring. Second, not only one auction will be considered, but also a series of auctions will be used to achieve an optimal solution.

1.2.5.1 Looking into the Future

The class `org:Thing` can be used to host the agents representation of the state of the world in a very general way. Since this work has an economic context, the state of the world is determined for an agent by its ownership and the class `org:Thing` is used to represent this ownership. Three types of such an (economic) Knowledge Base have to be distinguished: current KB, hypothetical KB, and a KB representing the difference between the former and the later.

The representation of the amount of things currently controlled by the agent is modelled by the class `org:CurrentKb`, subclass of org:Thing; whereby each agent, participating in the dynamic allocation process described here, has to have exactly one instance of org:CurrentKb.

$$org : CurrentKb \subseteq org : Thing \qquad (A.12)$$

$$org : \mathrm{HypotheticalKb} \subseteq org : Thing \qquad (A.13)$$

Neglecting eventual productive capabilities of agents the change to an instance of the class `org:CurrentKb` is reduced to exchange with other agents; thus, to auctioning as it is discussed in the previous section. Likewise, the change to the current KB will be represented by instances of the class `org:Auction`.

However, to prohibit that an agent gives something away below its own valuation for it, the preference for the thing an auction is about has to be calculated. In other words, the difference a change in the current KB would cause to the utility level of an agent is needed to determine the policy. To accomplish this a hypothetical KB has to be constructed. To identify the hypothetical KB as such the class `org:HypotheticalKb` is introduced (axiom A.13), analogously to the class `org:CurrentKb`.

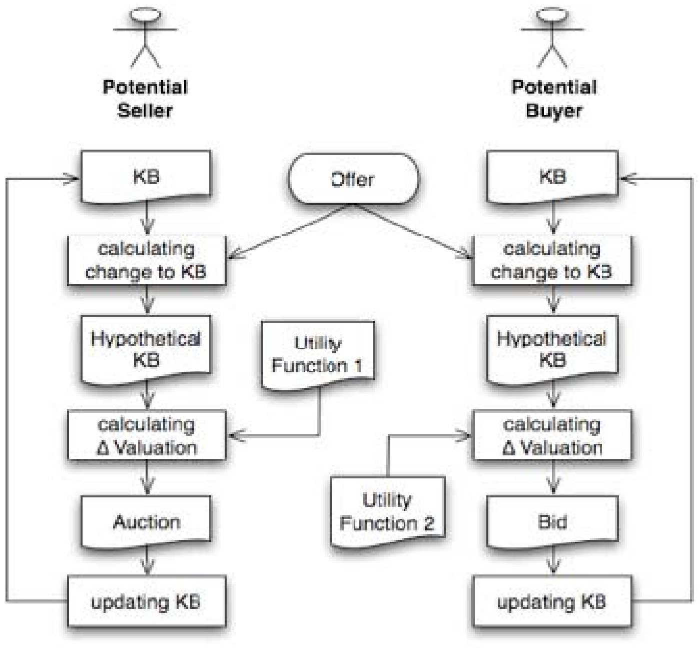


Figure 1.7

1.2.5.2 Determining the Value of Change

The calculation of the value of a possible action, leading to a change in the KB, can be reduced to the calculation of the difference between the current and the hypothetical KB, as well as the construction of the hypothetical KB.

The construction of a hypothetical KB, for a given current KB and a given change to it, depends on the schema of the KB. Thus, no general formula can be given, but a continuation of the example `org:Thing` used in the previous paragraphs illustrating the very steps of the construction of a hypothetical KB. In the head of the formula the current KB as well as the change, which will potentially be applied to it, are bound to variables. In the next step the difference is calculated. In the body of the formula the hypothetical KB is constructed and evaluated along with the current KB by the Utility Function of the agent.

```
{ ?currentKb org:Description {* :hasLiter ?lCur};
            rdf:type org:CurrentKb.
  ?change org:hasDescription {* :hasLiter ?lChange};
            rdf:type org:Auction
  ?lHyp sapl:expression "?lCur-lChange".
}=>{
  _:hypKB org:hasDescription {* :hasLiter ?lHyp}.
  sapl:I org:calculate {?currentKb ex:UtilityFunction *}.
  sapl:I org:calculate {_:hypKB ex:UtilityFunction *}}.
```

Likewise, this formula has to be implemented also for the agent on the buyer's side. However, constructing the hypothetical KB on the sum of the change, a won auction would bring, to the current KB.

The calculation of the value of the change to the agent is the calculation of the difference in valuation between the current KB and the hypothetical, as depicted in formula below.

```
{sapl:I org:calculate {?currentKb ex:UtilityFunction ?currentUtility}.
sapl:I org:calculate {_:hypKB ex:UtilityFunction ?hypotheticalUtility}.
?currentKb rdf:type org:CurrentKb.
?hypotheticalValuation sapl:expression "?hypotheticalUtility-?currentUtility".  ?change rdf:type
org:Auction.
}=>{?change org:hasValuation ?hypotheticalValuation}.
```

1.2.5.3 An Application Example

In this final paragraph it is presented how the approach can be used to dynamically distribute water between agents in such a way that a global utility maximum is reached.

The need of an agent for a share of the globally constraint resource (water), is expressed by the Utility Functions of the agent. Conforming to standard utility theory the Utility Functions are concave so that the first derivative (i.e. the bidding price) decreases with increased share of the resource. Thus, the second derivative of the Utility Functions is negative. Beginning from the Utility Functions and the initial distribution of the resource the market mechanism settles the distribution of the water. The goal is to maximize the global utility U_{glob} of all agents that they have from the resource water R. Whereby the global utility is the sum of the utilities the agents have $u_{glob}(r_1,...,r_n) = u_1(r_1) + ... + u_n(r_n)$, and the sum of water is the globally constraint by $R = r_1 + ... + r_n$. The global utility maximum is reached if the so-called Kuhn-Tucker conditions are fulfilled; thus, if all first derivatives of the agents Utility Functions are equal to a shared value. In other words, an optimal distribution of energy is reached if all agents express the same need (bidding price) for an additional unit of the resource.

In the figure (Figure 1.8), the performance of a simulation is shown. Three agents are assigned with initial share of the available water of 1.5 L; one agent is assigned with 3.5 L. From such an initial, not necessarily optimal, distribution equilibrium is reached after one round. A round is finished after each agent has enacted the auctioneer role. In the second round an agent tries to reduce the amount of water it holds by changing its Utility Function and consequently having a lower valuation for a share of the resource. Again, the system settles in the next round.

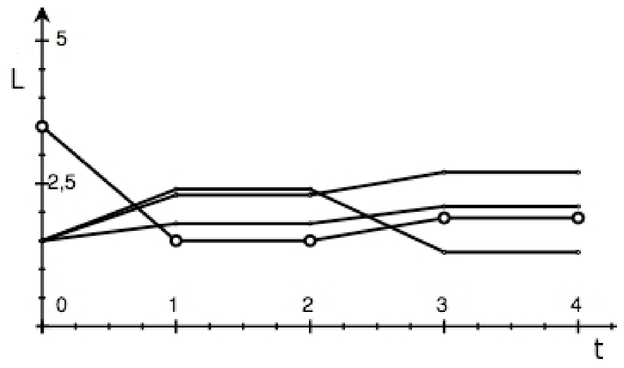


Figure 1.8

1.3 Observation in Multi-Agent Systems

This chapter focuses on the issue of observation in multi-agent systems (MAS). The term observation can be used to denote both the act of observation and the state of being observed. In the context of MAS the act of observation is clearly part of the perspective of an agent. The agent, as an autonomous and pro-active entity, can use its observational capabilities in order to achieve some goal or complete a task by taking advantage of the perceived information. We argue however that state of being observed should not be the concern of an agent, rather the environment it inhabits. More importantly the observability should not be entirely subjective decision of the agent, especially in competitive or open environments. Let us consider for instance a hostile agent that enters the MAS, observers cannot rely on it to make its suspicious actions observable. There is a call for component with predictable and configurable behavior that is in charge of managing each agent's observable state and behavior.

The proposed framework for Agent Observable Environment (AOE) builds upon the idea of explicit modeling of the environment in multi-agent systems, which was first put forward by Weyns et al in (Weyns et al., 2004). In the article, they argue for the importance of recognizing the environment as a first class entity in MAS with its own clear cut responsibilities, apart from agents. This is quite a different approach compared to the implicit and ad hoc treatment of the environment found often in the MAS literature (Weyns et al., 2004). Providing observability is identified as one of the main responsibilities of the environment (Holvoet and Weyns, 2005) and the AOE is designed to offer just that. Whereas the environment is said to cross-cut the MAS (Platon et al., 2007), we argue that the observation cross-cuts the environment.

1.3.1 Agent Observation Environment

The Agent Observable Environment (AOE) is a framework that allows the modeling of indirect interaction in MAS through observable changes to the persistent state of the agent's soft-body. The term soft-body was introduces to the MAS context by (Platon et al.,2005). In the context of AOE, soft-body can be thought of as an agent specific repository, which is modeled as part of the agent, but is managed by the environment. Figure 1.9 illustrates the logical architecture of AOE using the MAS layers as presented by (Viroli et al, 2007). The figure shows the two top most layers of the MAS. In addition to MAS middleware layer, the execution platform includes also the software deployment context that consists of operating

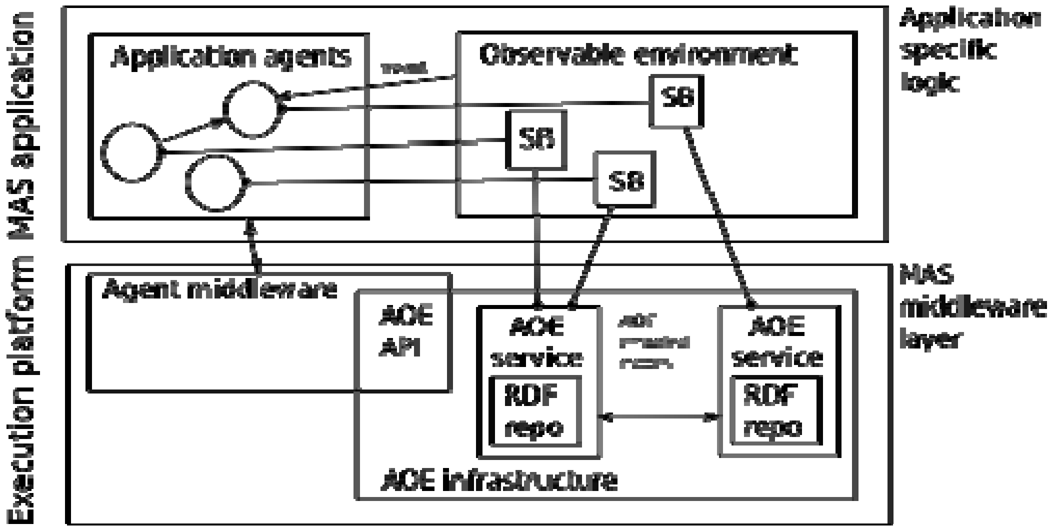


Figure 1.9 - The top application layers of AOE.

systems, virtual machines and other middleware. The bottom most layer is the physical support, which is divided into hardware and physical world sub layers.

Observable environment is basically the sum of all the soft-bodies in MAS and can be distributed across the nodes of the agent platform. The dotted lines in figure represent an inclusion relationship between the two objects. For example on the application layer the soft-bodies (SB) are modeled as part of every agent. Soft-body is created when agent is born and removed when the agent is killed. On the middleware layer the soft-bodies are managed by the distributed set of AOE service instances. When an agent migrates from one node to another, its soft-body must also migrate with it. If there is no AOE instance running on the target node, the agent must be informed about the situation before executing the migration. If the agent decides to migrate to a node with out AOE service, its soft-body is removed.

AOE uses RDF as the storage format for soft-bodies. RDF was chosen because of its status as the lowest common denominator between different semantic web (Berners-Lee et al., 2001) technologies. The requirement for successful utilization of AOE is, that the observing agents understand the sensed information at least on some level. For more sophisticated agents this might mean using ontologies and complex inference mechanisms, whereas the simpler agents might be hard coded with the logic that handles one very specific task based on its observations. Another benefit for using semantic web technologies is that is allows AOE to use inferred information when matching events related to the state changes with the observer's current interests. This kind of behavior naturally requires the use of ontologies. For example an observer might be interested to receive an observation related event whenever an object of class alarm or any of its subclasses is created to the AOE. With a proper ontology in place, the AOE is able to notify the observer when for example a printer alarm or house alarm is added. Assuming of course that both of the more specific alarms are modeled in the ontology as subclasses of the alarm the observer was interested in. Finally the event arrow from observable environment towards one of the agents represents a manifestation of observation i.e. an event caused by state change in AOE that matched the observation configuration of the agent receiving the event.

1.3.2 The main concepts of AOE

This chapter gives a more detailed description of the ideas behind the AOE including the soft-body, the act of observing, the use of ontologies and observation related rules.

1.3.2.1 Soft-body

A newly created soft-body is always empty. At runtime, both the agents and the environment can modify the content of the soft-body.

As noted earlier, the soft-body represents the observable state of an agent using RDF statements. Soft-body can include both state and behavior driven information. Just like in our physical world, every external action (as opposed to the activities going on inside the agent) induces a state change in the observable environment. The approach is similar to the Ferber's (Ferber, 1999) model of influences (agent actions), which are agents attempts to change the state of the world, and reactions, which are produced by combining all the influences of all the agents and local state and rules of the environment that finally result in state changes of the environment. In AOE, the state of the observable environment (OE) is changed as soon as the agent starts the execution of an action. For example, if the agent decides to start to sing, the action (agent X is singing) is reflected to the state of the OE. The implementation of the singing action can for example use the soft-body to make the lyrics of the song the agent is currently signing available for observation. The bottom line is that the description of the act itself is separated from the possible changes to the OE caused by that action, but they can both be observable.

Agent is only capable of changing the state of its own soft-body. It can receive information about the state of other soft-bodies in the MAS through observation, but is unable to modify that state. The selected model makes the handling of concurrent actions simpler, because the implementation must only manage the modifications made by the agent that owns the soft-body. If the agent is implemented as one thread, there is no need for synchronization. Environment based modification of the soft-body allows the designer of MAS to make selected action or state always observable. For example for BDI based agents, the designer of the AOE could make all the beliefs matching certain template automatically part of the agent's soft-body.

The observable actions in AOE are divided into two categories: practical and communicative. In order to understand the fundamental difference between these action types, we must first make a distinction between communication and mere interaction. Communication is defined as goal driven behavior that is aimed at informing the receiver of some fact or for creating the desired communicative effect, i.e. communicate presumes intention (Tummolini et al., 2005). Example of communicative action in MAS context is the sending of an ACL message, or in other words, committing a speech act (Tummolini et al., 2004). But interaction does not always have to be communication (Tummolini et al., 2005). Interaction can be mere transfer of data that causes a non-intentional state change to receiver. The receiver Y is the observer of some event, which is created as a result of some action performed by agent X. Agents X and Y are interacting indirectly via AOE, but not necessarily communicating. The agent X might not even be aware of being observed by the Y. The actual modeling of actions and choosing between the communicative and practical actions is an application specific task. This brings us to the subject of ontologies.

1.3.2.2 Ontologies

The original idea was to use one globally shared ontology to model the observable environment and allow agents to add only modeled entities to their soft-body. The application specific ontology would have been created by extending the AOE core ontology. This “ontology first” kind of approach, would have provided up to date model for the observable environment at all times and steered towards more well structured use of AOE by removing the possibility of ad hoc usage. However, the flexibility was chosen to be more important feature, so we decided to let the agents and their designers make the decision whether or not to use an ontology.

The current model includes very few restrictions on the content of the soft-bodies beyond the capabilities of RDF. The only restriction is, that only environment is allowed to modify concepts of the core ontology. The core concepts are illustrated in figure 1.10.

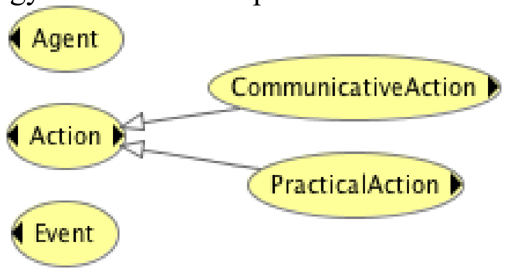


Figure 1.10 - Concept of AOE core ontology.

In addition to the classes, the ontology defines two properties whose domain and range are described in table 1.1.

Table 1.1 - Core properties of the AOE.

Property	Domain	Range
aoe:does	aoe:Agent	aoe:Action
aoe:observes	aoe:Agent	aoe:Event

The property aoe:does represents the ongoing action performed by some agent. For example, let us assume that migration of mobile agents is modeled as an action. When agent X starts its migration the statement X aoe:does :MoveAction is added to the X's soft-body. When the migration is completed the statement is removed. aoe:observes property is used to describe the observational interests of an agent. It is basically just a short cut for describing observation action.

1.3.2.3 Observation

If agent wants to observe something it must explicit configure the AOE according to its needs. The observation ontology (Viroli, 2001) defines two kinds of observation, static and dynamic. Static observation is suitable for a situation where the agents needs to know the current state of the environment. For example, the agent could check that the status of the printer is idle, before sending the document for printing. The result of a static observation is a snapshot of the observed state. If the agent is instead interested in the changes occurring in the observable environment, it can observe it dynamically. Dynamic observation means that the agent is informed whenever an event matching agent's configuration occurs in AOE. Possible event types are related to the adding and removing of data. Continuing the example where the agent is observing the status of a printer, the agent could, in case the status of the printer is busy, observe the status dynamically and be informed when it changes back to idle. In case of a static observation the observation configuration is removed from the AOE as soon the observation events matching the configuration have been made available to the observer. Configurations for dynamic observation are active until they are explicitly removed by the agent.

In the MAS literature the configuring of the environment is referred as setting the foci (Weyns te al., 2003) or creating a view (Schelfthout and Leuven, 2005). The dynamic configuration of the AOE allows agents to focus its observational capabilities to accommodate the task at hand. This does not however mean that an agent can receive information about any state or state change in the AOE. The agent's capabilities to observe can be limited by rules enforced by the environment.

1.3.2.4 Observational Rules

Rules are used in AOE to limit the visibility of information stored in the observable environment. They can be seen as controlling the reading rights of the information stored in AOE. Observational rules do not apply any restrictions on the information stored in the AOE nor can they be used to modify its state. Every rule has a level and a priority. The so called "laws of nature" or environmental rules, are the rules added by the designers of MAS. These first level rules always take the precedence over any other rules, but there is currently no sophisticated way beyond the use of rule priorities to handle conflicting rules of the same level. The second level rules are the ones added by the agents. Both rules can be added dynamically at runtime. The environmental rules and rules added by the agents are concrete tools for implementing objective and subjective coordination (Omicini and Ossowski, 2003) respectively.

The evaluation context for the rules is the content of the observer's soft-body. So for example, the rule can limit the visibility of the files made observable by agent X, to the agents that can be observed to be part of the same organization as agent X. Provided that the organizational information is part of the soft-body. We acknowledge the security issues related to the current arrangement. If the access to the files is only related to the value in the soft-body, it is easy for the observing agent to change that value on its own soft-body to something that grants the access. One way to improve the situation would be to make it possible to model some properties to be only modifiable by the environment and then create environmental rules for modifying those properties.

1.3.3 The prototype

This chapter describes the implementation details of the software component that aims to bring the features of the theoretical framework described in the previous chapter available to the designer of agent based applications. The implementation is at the prototype phase and does not at the moment implement all the functionalities envisioned by the theoretical framework.

1.3.3.1 General architecture

JADE was selected as the basis for implementation because it is the platform of choice for the UBIWARE project. It provides excellent facilities for extending the platform at both agent and kernel level. Developing an application with JADE usually involves the specification and development of set of agents, their behaviors and the relationships amongst them. The AOE implementation could include for example a set of lower level agents with roles for intercepting information, query processing and match making. However, since the role the AOE is more infrastructural one, the most natural place to implement it is at the execution platform level, not at the agent level. In this way the implementation also follows the logical model. AOE is part of the medium through which agent interaction is mediated. JADE framework offers convenient way to extend the platform with new kernel level services, thanks to its aspect inspired architecture.

With the version 3.1, the JADE moved away from the old monolithic and not so extendable kernel to the current service-oriented one. The new kernel architecture is based on the distributed version of the composition filter pattern. The idea behind the original composite filter is based on concerns and filters. Concern is the primary behavior of the program entity. The crosscutting concerns that extend and enhance the primary behavior are represented by filters. Filters can be used to inspect and manipulate both incoming and outgoing messages to the program entity. JADE extends this model by allowing concerns to be implemented in distributed manner.

The main components of a distributed kernel service are filters and sinks for incoming and outgoing messages, the slice, and vertical and horizontal commands. When an agent invokes a method in the service interface, the actual service can implement the operation directly or encapsulate the information about the method call into a vertical command and forward it to outgoing filter chain. Issuing a vertical command means that the actual implementation of operation is delegated to the outgoing sink. Before hitting the sink, the message travels through series of filters that can inspect, modify or even interrupt the vertical command. Any service activated in the node can provide its own filters for handling vertical commands issued by any other service in the node. In case of distributed service, where the affected components of the operation can exist on different nodes, the sink can transform the incoming vertical command into horizontal command and send it to the other nodes for processing. Horizontal command coming from a remote node is first received by the service slice. The slice can again implement the operation directly or delegate the execution to the target sink by creating and incoming vertical command and sending it through the incoming filter chain. An example of service implementation is illustrated in Figure 1.11, which depicts the route of an ACL message from one agent to another (Bellifemine et al., 2007).

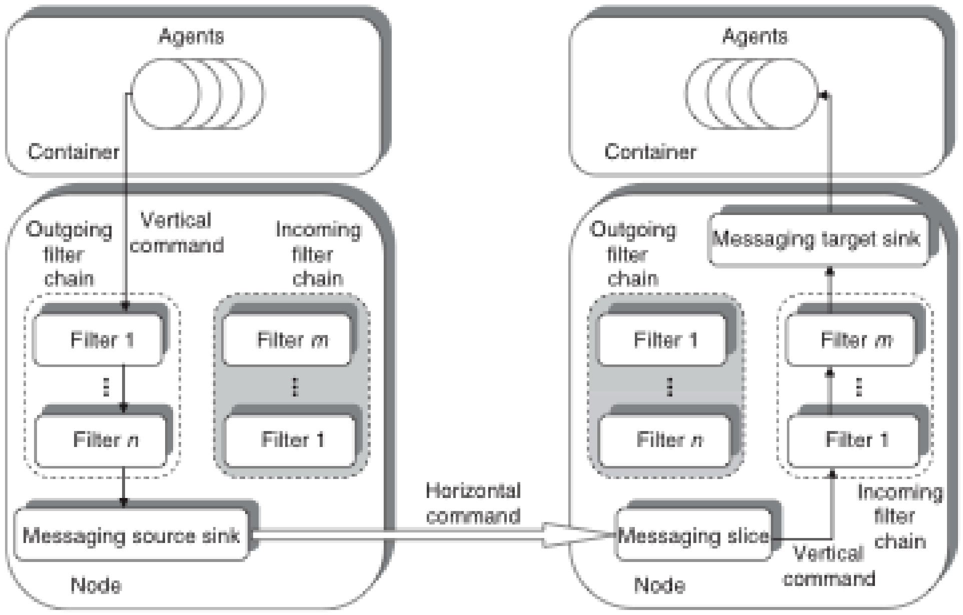


Figure 1.11 - Complete example of ACL message's route from one container to another.

In addition to the essential services like agent life-cycle management and messaging, the JADE distribution includes service implementations for event notification and topic based group messaging. Building your own distributed kernel service is relative simple using the base classes provided by the JADE framework. The main components needed for service implementation and their relationships are shown in Figure 1.12.

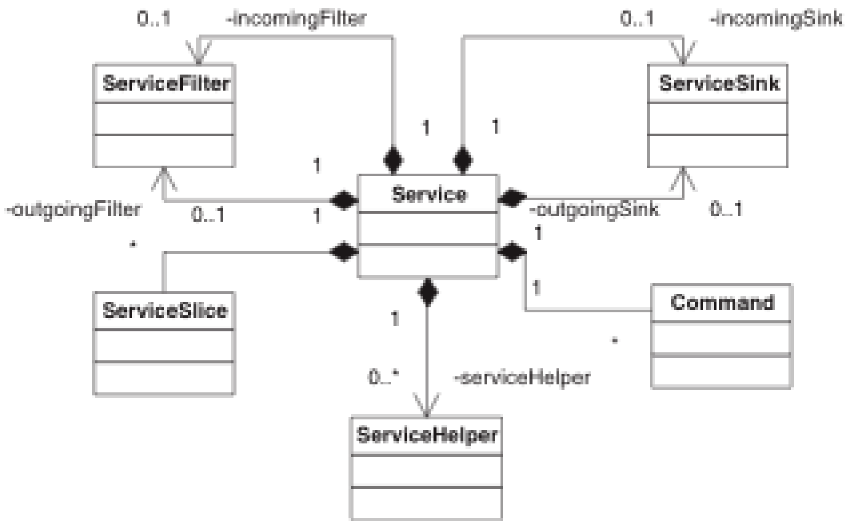


Figure 1.12 - Components of JADE kernel-level service [1].

Implementing AOE as an infrastructural service allows us to get a lower level access to the JADE platform using filters to the existing core services. First of all this makes is possible to design the framework to be relative transparent to the agent designers and minimize the work related using the AOE framework, so the agent does not have to worry about explicitly using the service, but still be observable.

Working at the kernel level makes it also difficult for agents to by-pass or misuse the service. This assures equal treatment of all the agents in both cooperative and competitive environments. It does not however mean that there can't be privileged agents in the system, which could for example modify or add new observational rules to the system.

One downside of service base implementation that might at first feel like major obstacle, is the fact that the service cannot be added to a running node. This means that the service must be started at same time as the platform node. With agent based solution, the starting of the service would be as easy as starting up a new agent. We don't consider this to be a major drawback. It is still possible to add AOE enabled nodes to the existing platform without shutting it down. We could for example start a new JADE instance with AOE, join it to the existing platform and move all the agents to the new instance.

Another implementation option would have been to use aspect oriented programming, for intercepting method calls related to observable information. This would have also fitted perfectly with the idea of observation being a crosscutting property of the environment. However, AOP was not chosen for implementation technology for the first prototype mostly for the same reasons as identified by the developers of JADE. In an effort to keep things simple and applicable also on mobile devices, the solution was crafted using JADE's built in extension mechanisms.

1.3.3.2 Implementation details

AOE is designed to be distributed as JADE plug-in. Figure 1.13 shows the technical architecture of AOE in relation to the other services existing on the JADE platform.

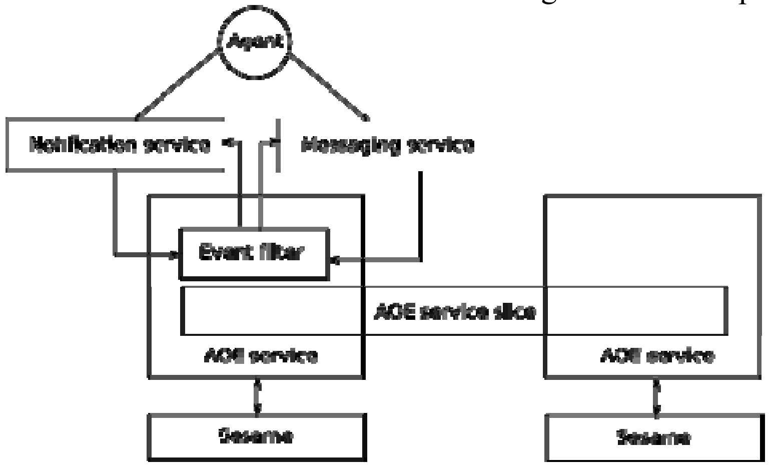


Figure 1.13 - Overview of technical architecture of AOE for JADE.

The AOE service uses information provided by the notification and messaging services. Filtering the commands issued by the messaging service, allows us to make the ACL messages observable to the possible eavesdroppers. It is not technically feasible to store all the ACL messages to the Sesame repository as suggested by the theoretical model. The current implementation does not actually store any of the messages, but this does not prevent the ACL message to be observed, because the event is generated at the AOE level.

The observation of practical action is currently limited to the JADE behaviors. This means that agent can observe what behaviors other agents are running. A behavior can become observable in two different ways. First of all, the agent can use the AOE service to intentionally make its actions observable. Another option is use the AOE service configuration to list the names of the observable behaviors and system will automatically make them observable as soon the agents adds it to its scheduler. The configuration based solution is implemented using events of the notification service to keep track of the behaviors running on each agent. This notification behavior is not enabled for the agent by default, because the amount of events even a single agent can create. The processing of large amount of events naturally has a negative impact on the overall performance of the platform. In the current implementation this problem has been at least partly solved by modifying the base class for all the JADE agents, by adding a new private boolean property called generateObservableBehaviorEvents. By setting this value to true, the agent will only generate behavior events relevant to the AOE. In other words the events are only generated when a behavior is added or removed. The solution is a bit of a hack, but it was very easy to implement and it has no side effects to the normal behavior of JADE. It is also possible to improve the performance by interrupting the notification events. This solution is only viable when there are no other services or agents that need to use the events.

We use the Sesame 2.2, an open source RDF repository distributed by Aduna software under Aduna BSB-style license (see http://openrdf.org), for storing the state of the observable environment. The soft-body of an agent is implemented as a context in Sesame. When agent X adds the property currentTemp with value 30 to its soft-body, the statement “X currentTemp 30” is stored to the context X. using Sesame.

We will next zoom in to the inner workings of the AOE service. The diagram X, shows the main classes of the AOE service.

1.3.3.2.1 Main classes of AOE implementation for JADE

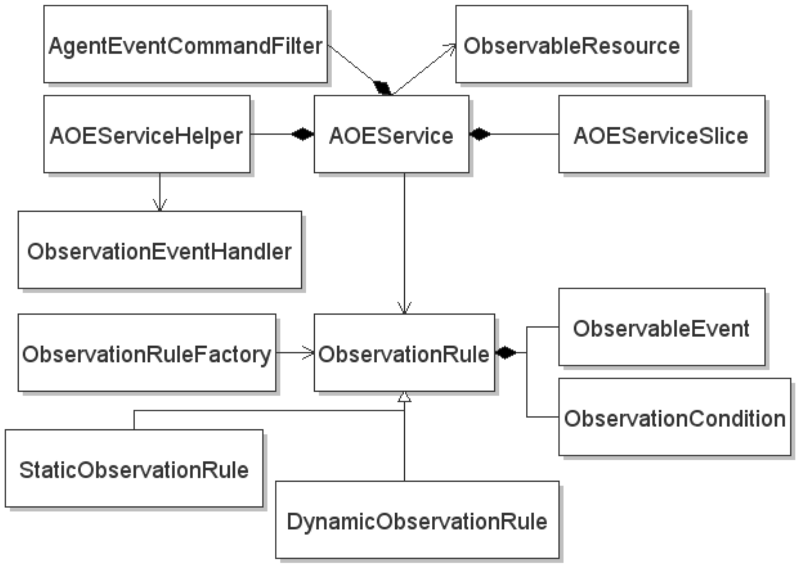


Figure 1.14 - Design phase class diagram of the AOE service.

AOEService Responsible for booting and configuring the service. Handles the connections to the underlying RDF repository and provides methods to matching event to the observation configurations.
AOEServiceHelper The interface through which the agents use AOEService. Provides methods for soft-body management and static and dynamic observation. Implementation can be retrieved from the base of class of JADE agent by calling getHelper() method. Every agent has its own helper instance that works as the event queue for AOE. The event queue is processed by a simple CyclicBehavior implemented as an inner class of the helper, responsible for retrieving and executing the event handler associated with the observed event.
AOEServiceSlice The part of distributed AOE that receives incoming horizontal command from other nodes running AOE.
ObservableResource Wraps the data of an RDF individual. Implementation is based on LazyDynaBean that is part of the BeanUtils package distributed by the Apache Commons project. DynaBean is an object that support properties whose names, data types and values can be dynamically modified. ObservableResource class provides object oriented way to manage the agent's soft-body. For example when simple reactive agents, the developer can use this class to interact with the soft-body without any knowledge of RDF.
ObservableEvent Represent a RDF statement with context. ObservableEvent also includes the type of the repository event (add/remove) from which the statement originated.
ObservationRule Observation configuration of an agent. Rules come in two flavors for static and dynamic observation.
ObservationEventHandler Interface that must be implemented by the classes that handle the results of dynamic observation. Contains only one method handleObservation(Set<ObservableEvent> events).
AgentEventCommandFilter Incoming vertical command filter for handling event from the messaging and notification services. Currently the list of filtered events is as follows: 1. Agent created 2. Agent killed 3. Agent state changed 4. Agent moved 5. Behavior added 6. Behavior removed 7. ACL message sent

1.3.3.2.1Optimizing performance of frequently updated properties

The design decision to use RDF and the only format for storing soft-body data has raised questions about the overall performance of the approach. This is a valid concern especially if we consider the case where frequently updated property is being dynamically observed by a group of agents. The described situation puts both the repository and the AOE service under a lot of the stress. AOE service must propagate massive amount of events through out the possibly distributed platform and the repository must perform both remove and add operations every time the value of the property changes.

We have come up with couple of designs that should alleviate the problems related to frequently updated properties at the repository level. The basic idea is to use different types of repositories for different types of properties. The less time critical data can be stored to the Sesame repository, while the frequently updated properties can use for example memory based relational database. The type of the property can be configured to the system by identifying the repositories using namespaces. If the namespace solution is unusable because of legacy data for example, it should also be possible to configure repositories for individual properties. If no configuration is found for certain property, it is stored using the default repository.

Using separate repositories is relatively straight forward when it comes to adding and removing or information. Things get a little bit more complicated for the querying though.

The simplest solution is to allow agent to query (observer) only one repository at a time. There are mapping tools like SquirrelRDF (http://jena.sourceforge.net/SquirrelRDF/) and D2RQ (http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/) that allows non-RDF data store, like relational databases and LDAP servers to be queried using the SPARQL or Sesame API. (see Figure 1.15). It should be noted though, that both of the introduces tools have their limitations when it comes to the types of queries they can handle. We also don't know, if the data can be used for inference.

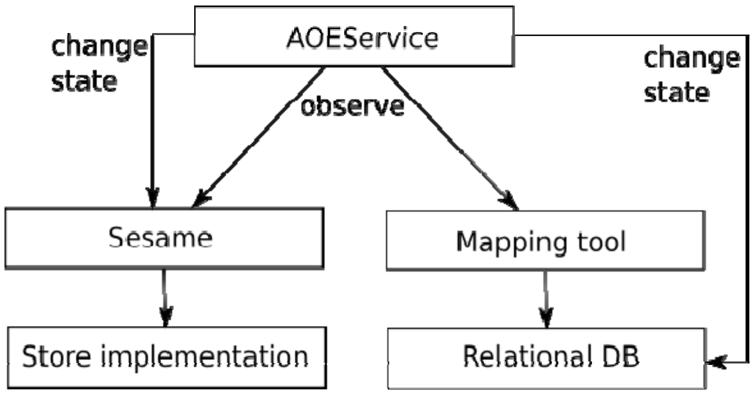


Figure 1.15 - Optimizing performance using different types of repositories.

Total separation of repositories might be a reasonable solution if observation can be done with out any conditions, because the conditions are also evaluated in only one repository at a time. For example if the currentTemp property is stored to a MySQL database and the name of the agent in Sesame repository, it would not be possible to condition the observation of the currentTemp property with the name of the agent. As a matter of fact, if the currentTemp was the only property stored in MySQL database, the observers would always observe all its available values. Using multiple repositories can also make it impossible to use inferences for observation, because there is no global view of the data available.

In order to make the usage of multiple repositories a viable solution for complex application, we need to be able to distribute the query processing. There are distributed RDF repositories out there (Cai and Frank, 2004), but they are not the solution, because using RDF only repositories brings us back to the original performance problem of frequent updates. DARQ is an engine created by Quilitz and Leser (Quilitz and Leser, 2008) for executing federated SPARQL queries against multiple data sources in a totally transparent way. Using a component like DARQ it is possible, at least in theory, to create data access layer for the AOE that combines the performance of traditional relational databases with flexibility of RDF. The Figure 1.16 shows the proposed architecture including distributed query management

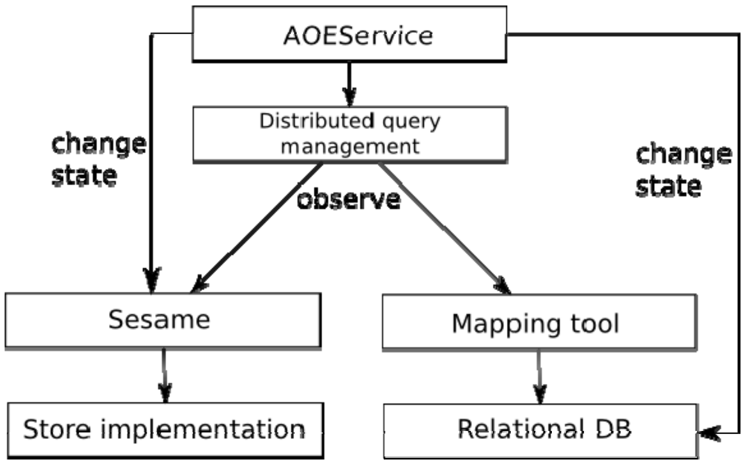


Figure 1.16 - Optimizing repositories using multiple repositories with distributed query management.

1.3.3.2.2 Comparison to the theoretical model and further development

The current prototype version includes the basic functionality to implement application modeled using agent observable environment framework The most important features still missing or in their infancy are related to the observational rules and conditions, the range of observable action types and efficiency of the distributed AOE.

There is currently no way to limit the visibility of the data published to the agent's soft-body. The appropriate place for adding the rule processing is the element that compares the events created by the agents' actions to the observation configurations. In addition to the context of the soft-body there might need to use platform specific information for the evaluation of the rules. For example for performance reasons, the visibility of certain properties can be limited to the agents residing at the same node as the source.

The simple implementation provided for observing practical actions, might be too coarse for more complex applications. The execution of a JADE behavior can include nested actions like invocation of remote services using techniques like web services, RMI or plain HTTP and of course calls to the methods of the local service components, which might be just as valuable targets for observation. Using the current filter based approach, the sources of the action related information are limited to the services of the JADE platform. The next step in extending the selection of observable actions could be the use of some AOP framework..

Then the developer of the AOE could bind the observable events to individual methods. There are AOP solutions available that modify the compiled byte-code in JVM and allow you to dynamically add aspects to already loaded classes. No source code or recompilation is needed. That would mean that the observable description of the actions could be totally separated from the actual implementation.

Finally there is the issue of distributed AOE. The current mechanism for propagating the observable environment to other nodes is based on complete replication of the state. This is hardly the most efficient solution. The replication strategy could be improved by dynamically configuring the data to be replicated using observational rules on each node. For example if there are no agents observing property X on a remote node Y, there is no need to send state change events about X to that node. Different kind of optimization could be implemented by making the agents or their clones to move closer to the entity they are observing. If the agent X wants to observation the property Y on a remote node Z, the observation would cause the agent X, or its clone, to automatically migrate to the node Z.

1.3.3.3 UBIWARE integration

Because the JADE implementation of the framework is distributed as plug-in for the standard JADE library, some integration work is needed in order for it to be useable by the UBIWARE agents. This chapter describes two main approaches for using AOE from UBIWARE based agents. The first one is based on RABs and the second one introduces some AOE specific changes to the UbiwareAgent class. The advantage of the first alternative is that it makes use of the common extension mechanisms of the UBIWARE platform and therefore creates no tight coupling between the agent and the new service. The deep integration in the latter case however offers more possibilities for control over the way observable information is included to the agent's runtime cycle and for performance tuning. In general, the integration consists of two parts: The description of the S-APL code needed to interact with the RABs and the java code.

1.3.3.3.1 Creating new RABs

Reusable atomic behaviors are the basic building blocks for implementing any agent-environment interaction in UBIWARE. We present three new RABs that allow UBIWARE agents to observe the environment and modify their soft-body.

In addition to the RABs a generic observation handler implementation is needed. The handler is responsible for making the data of the events part of the agent’s beliefs. In AOE the result of an observation, both static and dynamic, is always collection of ObservationEvent instances. ObservationEvent class provides method toRDF(), which returns RDF representation of the event. The implementation of the handler then becomes rather trivial task of iterating over the received events, transforming them to RDF and adding the statements to the agent’s knowledge base.

StaticObservationBehavior

Action:

Queries the AOE for statements that matches the template.

Inputs:

Name	Meaning	Mandatory	Default
isAsync	Whether the method is executed asynchronously or not.	No	false
RDFTemplate	RDF statement template that is used to query the agent's observable environment. For example * :currentTemp *. Instances can be queries using property rdf:type.	Yes	
context	Can be used to limit the query to certain context (=agent) only. If no context is used, the whole observable environment is queried.	No	
condition	Can be used to filter the matching events. Condition is given in SPARQL. The variables used in the condition must match the variables introduced in the rdf template. See the examples.	No	
includedProperties	List of properties included when querying based on rdf:type. By default, all the properties are returned. This parameter has no effect if the queried property is not rdf:type.	No	

Outputs:

Set of beliefs created by filling in the variables in the RDFTemplate parameter are added to the agent's KB.

Example of usage:
Retrieving all the agents with current temperature more than 30. Please note, that if the agent has currentTemperature over 30 as part of its own soft-body, it will also be part of the returned statements.

```
{ sapl:I sapl:do java:StaticObservationBehavior }
sapl:configuredAs {
     p:RDFTemplate sapl:is {?x :currentTemp ?z} .
     p:condition sapl:is {
          FILTER(?z > 30)
     }
}
```

Example of output:

```
X1 :currentTemp 40
X2 :currentTemp 50
```

Example of usage:
Retrieve all statements related to currently observable alarms.

```
{ sapl:I sapl:do java:StaticObservationBehavior }
sapl:configuredAs {
     p:RDFTemplate sapl:is {?x rdf:type :Alarm}
}
```

Example of output:

```
Alarm1 :type Alarm .
```

```
Alarm1 :message “Out of paper”
Alarm1 :source Printer1
Alarm2 :type Alarm .
Alarm2 :message “Paper jam”
Alarm2 :source Printer2
```

Example of usage:
Retrieve all alarms associated with the agent X1.

```
{ sapl:I sapl:do java:...StaticObservationBehavior }
sapl:configuredAs {
     p:RDFTemplate sapl:is {?x :hasAlarms ?z}
     p:context X1
     p:includedProperties sapl:is {
          :predicate :type
          :predicate :source
     }
}
```

Example of output:

```
Alarm1 :type Alarm .
Alarm1 :source Printer1 .
Alarm2 :type Alarm .
Alarm2 :source Printer2
```

DynamicObservationBehavior
Action:
Configures the AOE to notify the agent when the configured event happens.
Inputs:

Name	Meaning	Mandatory	Default
eventType	possible values: add, remove	Yes	
RDFTemplate	see StaticObservationBehavior	Yes	
context	see StaticObservationBehavior	No	
condition	see StaticObservationBehavior	No	
includedProperties	see StaticObservationBehavior	No	

Outputs:
RDF description of the observable event.
Example of usage:
Agent will receive notification whenever some agent in the organization ORG1 executes InterestingAction.

```
{ sapl:I sapl:do java:DynamicObservationBehavior }
sapl:configuredAs {
     p:eventType “add”
     p:RDFTemplate sapl:is {?subj :does :InterestingAction} .
     p:target sapl:condition {
          <http://ex.com/ont#ORG1> :organization ?subj
     }
}
```
Example output:
```
X1 :addedStatement { X1 :does :InterestingAction }
```
where X1 is the context in which the change occurred i.e. the agent whose soft-body was changed.

Example of usage:

Agent wants to receive notification whenever an alarm is added to the AOE.
```
{ sapl:I sapl:do java:...DynamicObservationBehavior }
sapl:configuredAs {
     p:eventType "add"
     p:RDFTemplate sapl:is {
          ?x rdf:type :Alarm
     }
}
```
Example output:
```
X2 :addedStatement {
     Alarm1234 :type Alarm .
     Alarm1234 :message "Out of paper"
     Alarm1234 :source Printer1
}
```

SoftbodyManagementBehavior

Action:

Is used to publish, remove and get information for the softbody

Inputs:

Name	Meaning	Mandatory	Default
action	Possible value: show, remove	Yes	show
target	RDF statements for which the given action is executed. _self can be used to refer to agent itself	Yes	

Outputs:

None

Example of usage:

Adding currentTemperature and alarm to the soft-body.
```
{ sapl:I sapl:do java:SoftbodyManagementBehavior }
sapl:configuredAs {
     p:action sapl:is show .
     p:target sapl:is {
          _self :currentTemperature 30 .
          _self :hasAlarm _x .
          _x rdf:type PrinterAlarm .
          _x :message "Out of paper"
     }
}
```

Removing all statements from the SB

```
{ sapl:I sapl:do java:SoftbodyManagementBehavior }
sapl:configuredAs {
     p:action sapl:is remove .
     p:target sapl:is {
          ?x ?y ?z
     }
}
```

1.3.3.3.2 Modifying the UBIWARE agent

This approach basically means that the AOE is made an integral part of any UBIWARE agent in a same way that support for GUI or HTTP events is in the current implementation.

Event approach:

The current version of the UBIWARE supports S-APL code that process events from GUI and network sockets through predicates sapl:eventFromGUI and sapl:eventFromSocket. This approach could be extended to work also with events from the AOE as well. The UBIWARE agent would store all the observed events to its blackboard and add the belief of the following form to its knowledge base:

```
     {…} sapl:eventFromObservableEnvironment {…}
```

Ontology based solution:

The UBIWARE agent would automatically recognize beliefs containing information available in the AOE. These belifies could be identified based on some configuration or, if an ontology is used, based on some parent type or annotation value. For example when statement “PM1 aoe:currentTemperature ?x” is added to the agent's knowledge base, the runtime cycle would notice that the property is from the namespace that contains only observable properties. Based on this information the agent could then initiate a static observation in order to retrieve the value of the property. Because of the dynamic nature of the AOE, it is possible that the requested property is not observable by the agent at the time of the execution runtime cycle. In such case, the agent could automatically configure the environment to notify it when the value becomes available. Based on the requirements, the agent can either continue to observe the property of remove the configuration as soon as the first dynamic observation has been received.

UBIWARE Deliverable D2.1:
Workpackage WP2:
Task T2.1_w2:
Workpackage leader: Sergiy Nikitin

2 UbiBlog – Managing Distributed Resource Histories

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

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

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

The problem of efficient data sharing, exchange and reuse within complex systems plays a key role for usability of the UBIWARE platform and its commercial success. In this workpackage we introduce a mechanism for distributed data management within the UBIWARE platform which allows platform users to build industrial business solutions.

2.1 Distributed Querying Scenario in Paper Industry

The scenario we have selected is based on the real software infrastructure from process industry. There is a complex production line (e.g. paper producing machine) which is served by a number of control and diagnostic systems. Those measurements taken from sensors that go beyond the range are stored in Alarm History Database. The Diary Database contains records about critical alarms and comments of the maintenance workers. There are also comments on actions taken. Also, there is a Scheduled Performance Monitoring database that stores results of the analysis which is performed once per day. The analysis includes all nodes with performance indices that indicate how well each node is performing (see Figure 2.1).

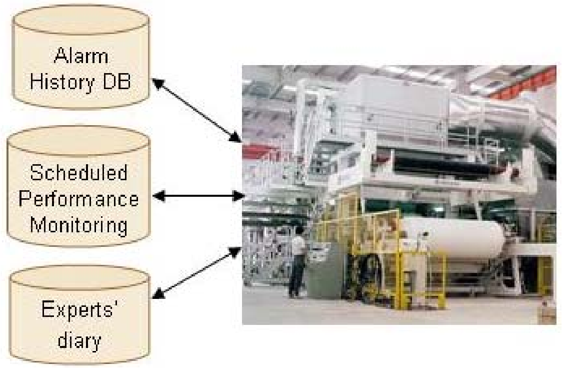


Figure 2.1 - The IT-infrastructure of the paper machine.

Suppose, there is a serious fault happened in the paper machine that has led to the subsequent alarming and maintenance actions. All these events are recorded in the respective databases. However, in order to analyze the preceding events, say during one week before the fault, an expert may need to query portions of data from all the databases, then change filtering parameters and query databases again and again. The easiest way is to provide an expert with the interfaces to all the databases separately, however, the expert's decision will be stored (if will be at all) to a separate file or another database. It will be hard to find it if similar problem occurs later. Thus there is no integrated view on all the contents of the databases, neither on other sources of information about paper machine operation. In Semantic Web domain it is called a proof when any knowledge is connected with the rules and facts that were used to infer it. The problem of integrated view on the information is really important in the industrial automation, especially due to the fact that a lot of experienced experts in a majority of companies are going to retire during next 5-10 years, without proper knowledge transfer to new experts. On the other hand, semantic linking of the information will be in a great demand when the standardization efforts taken in companies will go beyond the companies' boundaries and will call for a unified mechanism of distributed querying for knowledge and expertise exchange (Figure 2.2).

In the idealistic case, companies will be able to sell information and analytic services to each other seamlessly with small or no programming efforts. The services sold will be easily integrated into the company environment with the guaranteed compatibility. We, therefore, foresee the need for tools and capabilities in the UBIWARE platform that will simplify distributed querying and information integration.

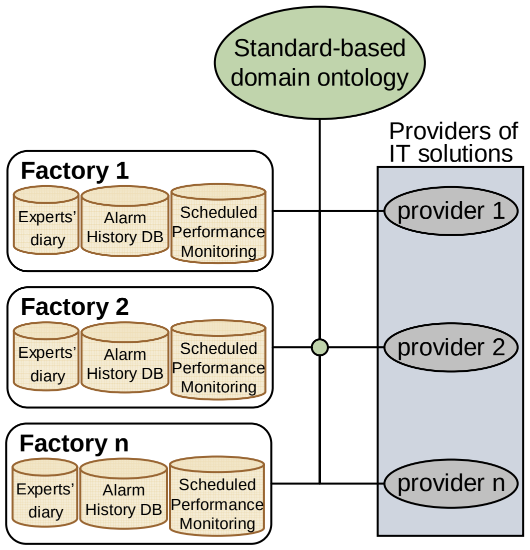

Standard-based domain ontology
Providers of IT solutions
Factory 1
Experts' diary
Alarm History DB
Scheduled Performance Monitoring
Factory 2
Experts' diary
Alarm History DB
Scheduled Performance Monitoring
Factory n
Experts' diary
Alarm History DB
Scheduled Performance Monitoring
provider 1
provider 2
provider n

Figure 2.2 - Inter-company standardization in paper industry.

2.2 ONTONUTS concept

We introduce Ontonuts concept to facilitate the presentation of modular scripts and plans in the UBIWARE platform. The idea is somewhat similar to the notion of Enterprise Java Beans, whereas it is semantic by nature and has its own architectural peculiarities. Instances of Ontonut concept in general represent a capability with known input and expected output. Then we adapt Ontonuts to the problem of distributed querying.

2.2.1 Ontonuts in the architecture of the UBIWARE platform

The Ontonuts technology is implemented as a combination of a Semantic Agent Programming Language (S-APL) script and Reusable Atomic Behaviors (RABs), and hence, can be dynamically added, removed or configured. Being run on the platform, Ontonuts can be understood as an engine or agent role. From the programming point of view Ontonuts allow componentization of S-APL code by introducing a semantic annotation to it. Such annotated pieces of code are called capabilities (analog of function in procedural programming). The capabilities further can be dynamically combined into plans and put to the execution by the Ontonuts engine.

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

$$Ontonut : \{script, precond, effect\} \tag{2.1}$$

The semantic annotation of Ontonut (by precondition and effect) allows us to automatically plan (compose) agent's activities to achieve a specified goal. The script part has an S-APL code that produces the effect based on the precondition. The whole data model of the UBIWARE platform is triple-based, therefore goals, preconditions and effects are defined as triple sets in S-APL. Suppose we have an initial data set {A A A}, a goal G1 defined as {C C C} and we have two ontonuts O1 and O2, defined as:

```
O1 :type :Ontonut
O1 :precondition {A A A}
O1 :effect {B B B}
O1 :script {{A A A}=>... =>{B B B}}

O2 :type :Ontonut
O2 :precondition {B B B}
O2 :effect {C C C}
O2 :script {{B B B}=>...=>{C C C}}
```

The appearance of the goal G1 will activate Ontonuts engine that will match G1 against available effects and apply planning, which will result in an execution plan: O1=>O2=>G1.

2.2.2 Ontonuts in a nutshell

The Ontonuts functionality is built on top of the platform and reuses available scripts and RABs without any modifications to the platform. Architecturally, Ontonuts engine consist of three main components: Triggering rule, Action Planner and Plan Executor (Figure 2.3).

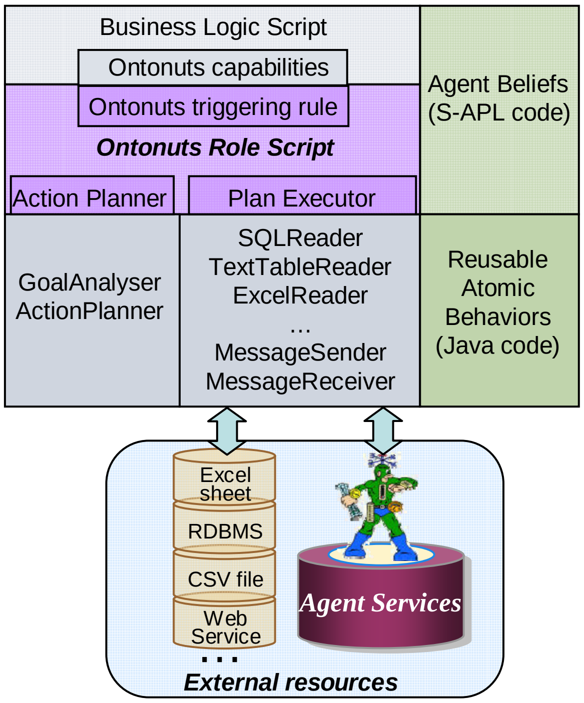


Figure 2.3 - Ontonuts architecture.

The Ontonuts Triggering Rule is a starting point of the engine work. The Triggering Rule is a MetaRule, i.e. it runs before other rules and commitments fire. On each iteration of *Live behavior* (the basic default behavior of every agent), the rule checks whether there are any Ontonut calls to be handled. Ontonut capabilities may include interaction with other agents or external resources such as databases, files or web services. On the other hand, capabilities can perform local actions and do some computations over the data, e.g. statistical analysis.

2.2.2.1 Invoking Ontonuts

The Ontonuts engine supports three types of Ontonut calls:

- Explicit
- Goal based
- Pattern-based

The Explicit call to the Ontonut has following syntax:

```
{sapl:I sapl:do ont:Ontonutid}
  sapl:configuredAs
{x:precondition sapl:is {Input
                        statements}
}.
```

The result of the call is added to the G – a general context. Instead of creating a container with input statements, it is possible to point to the existing one by container id. The content of the container will be treated as an input.

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

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

The Ontonuts engine runs the planner, to check if the plan can be produced for the goal taking into account the initial data provided. If not, then the goal is marked as non-achievable:

```
:id ont:isAchievable false.
```

If the planner comes up with the plan for the goal, it assigns a plan to the goal id:

```
:id ont:plan :planid.
:planid sapl:is {plan statements}.
```

The plan statements are ready to be executed within the agent's beliefs. The Ontonuts engine just puts them to the general context G, where they are run as any other commitments. Upon completion of the plan execution, the following statements are added:

```
:planid ont:isPlanAchieved true.
:planid ont:execResult {result
                        triples}.
```

In case of non-completion or failure the false value is assigned to the *ont:isPlanAchieved* property. However, if the plan has been executed successfully, but resulted in an empty set, the ont:execResult property may point to false value.

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

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

This call can be considered as a syntactic sugar for goal definition, where left part of the commitment is considered as a goal. The Ontonuts engine intercepts such commitment before it fires, then removes it from G and uses left part to perform planning and execution. However, this type of call does not specify the initial data set as the second type of call does. Such goal definition is possible for those Ontotnuts, which precondition is always true within the goal specified. For example, an Ontonut can perform queries over certain database and use a (sub-) pattern of the goal to produce a query and execute it.

After the goal is achieved, and, hence, the variable values in the left part of the commitment can be assigned, the Ontonuts engine produces the result (the right part) of the commitment using the variable values. If the evaluation of the left part results in an empty set, the variable values will not be assigned, so, it is recommended, that user defines sapl:else construct in the right part of the commitment to ensure alternative action:

```
  {A A ?a} => {some action with ?a };
              sapl:else {alternative}.
```

This type of Ontonuts targets mainly the tasks of distributed querying that are discussed in the forthcoming subsections.

2.2.2.2 Planning the execution

The planning is organized as a goal-driven process. We apply backward chaining algorithm to build an action plan, which may involve other Ontonuts. The planner does semantic inference over the set of initial data before the actual plan generation starts. The semantic annotations of Ontonuts, as well as corresponding domain ontology are, therefore, a key success factor of the Ontonuts-based applications. The planner acts in a straightforward way – it matches the goal against Ontonut annotations by subtracting (operation over sets) these annotations from the goal. If a goal can be fulfilled by available initial data and Ontonuts, the planner starts to check whether the preconditions of these Ontonuts can be fulfilled. If the preconditions may need to use other Ontonuts, they are checked as well. In such iterative manner, the planner builds a solution tree. The planner then may choose the preferable solution using different criteria, e.g. utility-based selection.

The formal definition of the goal-driven planner is given below:

Let the goal be defined as $Goal = \{T\}$ – a subgraph defined as a set of triples T. The triple is defined as a tuple $T = (s, p, o)$, where s, p and o are subject, predicate and object of the triple. However, we extend the range of s, p and o with the variable value, i.e. in any place of s, p and o there can be a variable $v = ?name$. Such extension introduces set-subset relationships amongst triples, e.g. $T_1(A, A, A) \subseteq T_2(A, A, ?a) \subseteq T_3(?x, A, ?a)$, which become sub-graphs. The Ontonut is a pair $O = (P, E)$, where P is a precondition $P = \{T\}$, and E is an effect $E = \{T\}$ of the Ontonut O. We also introduce an initial data set as $I = \{T\}$, which defines the facts. Before starting the planning process, a necessary condition should be met:

$$Goal \setminus (I \cup E_O) = \varnothing \tag{2.2}$$

where E_O represents effects of available Ontonuts. If the goal can be resolved with the initial data set, i.e. $Goal \setminus I = \varnothing$, then the planning is over and the result answer is given (a subset of triples from the initial set that match the goal). If there are Ontonuts, needed to fulfill the goal, we define a subset of the goal as:

$$G_0 = Goal \setminus I \tag{2.3}$$

and then introduce a set of solutions for the G_O goal:

$$S^{G_0} = \{S_i^{G_0}\}, i = \overline{1, m} \tag{2.4}$$

each solution S_i is a combination of Ontonuts such that:

$$\begin{aligned} &S_i^{G_0} = \{O_j\}, j = \overline{1,n}, \\ &G_0 \setminus \{E_{O_j}\} = \varnothing, \\ &S_i^{G_0} \neq S_k^{G_0}, i,k = \overline{1,m}, i \neq k \end{aligned} \tag{2.5}$$

where E_{O_j} - an effect of the Ontonut O_j. The equation (5) defines an obligatory condition for a solution to solve the goal and ensures uniqueness of each solution. Each solution has preconditions of its Ontonuts P_{O_j}. Union of the preconditions defines the subgoal of the solution (i.e. what data we need to make the solution work):

$$G_i^{G_0}(S_i^{G_0}) = \bigcup_{j=1}^{n} P_{O_j} \tag{2.6}$$

where $G_i^{G_0}$ - a subgoal of the i-th solution $S_i^{G_0}$ of the goal G_0. The goal-solution graph G can be defined as a set of pairs:

$$\begin{aligned} &G = \{G_i^{G_j}, S_i^{G_j}\}, \\ &\exists G_i^{G_j}(G_i^{G_j} \setminus I = \varnothing), \\ &\forall G_j(G_j \setminus E(S_i^{G_j}) = \varnothing) \end{aligned} \tag{2.7}$$

where $E(S_i^{G_j})$ is a union of effects of the Ontonuts involved in the solution $S_i^{G_j}$, i.e.

$$E(S_i^{G_j}) = \bigcup_{k=1}^{n} E_{O_k}, S_i^{G_j} = \{O_k\}, k = \overline{1,n} \tag{2.8}$$

The goal-solution graph contains a set of subgoal nodes linked to each other with *subGoalOf* relationship. Each subgoal may have a number of solutions. Each solution determines the way the subgoal can be achieved. If at least one subgoal $G_i^{G_j}$ can be achieved purely within I, so that $G_i^{G_j} \setminus I = \varnothing$, then the subgoal G_0 can be resolved.

The goal solution graph will result in a set of Ontonut calls, linked to each other. We distinguish five main types of call combinations (see Figure 2.4).

The first and second types are parallel and sequential ones. The third type is the combination of first and second, where output from a set of Ontonuts executed in parallel is used as an input for one subsequent Ontonut. The fourth type uses output of one Ontonut as an input for a set of subsequent parallel ones. And the fifth type displays the case when an output is used as an input for subsequent and non-subsequent, parallel and non-parallel Ontonuts.

The execution plan produced by the planner uses handlers to determine the status of the execution and trigger sequential steps upon completion of preceding ones.

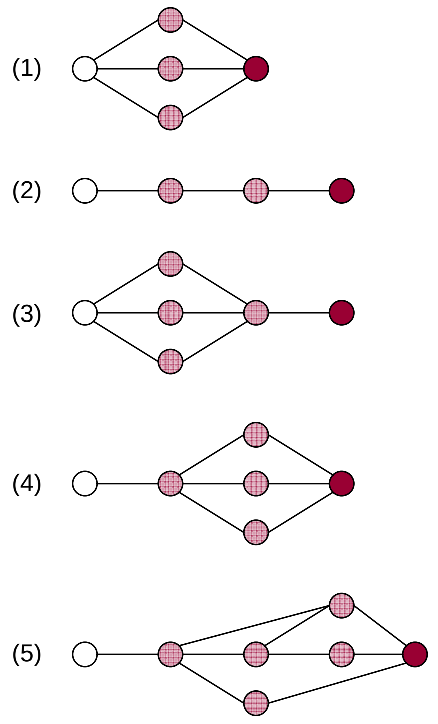


Figure 2.4 - Ontonut combinations.

2.2.2.3 Handling the execution

The Ontonuts engine does not execute the plan as a whole; it rather generates a plan that is run by the agent’s Live behavior engine. However, the plan is not straightforward; it includes additional handlers that allow the Ontonuts engine to observe the state of the execution and react if the execution can not be successfully completed. The plan is sequential and therefore it has steps or control points. At each control point the plan produces the statements that represent the status of the execution. These statements are collected in a container that is attached to the plan:

```
:planid ont:execStatus {
    :01 ont:status ont:Success.
    …
    :nn ont:status ont:NoResponse.
}.
```

The engine then can use the status information for re-planning if the current plan did no succeed.

There are two classes of Ontonuts that differ by the execution:
- Self-running
- Engine-running

The latter type has a built in script, that runs in agent’s Live behavior as an independent code and returns the result to the G container (to the agent’s active beliefs). The former is rather a description that is recognized and executed by the Ontonuts engine. The engine-running Ontonut calls are presented in the plan as explicit ones (see 2.2.2.1). In the current version, the engine supports only one type of engine-running Ontonuts, that provide access to the databases (we call them Donuts). More information on Donuts and engine-running Ontonuts will be provided in the sub-section 2.2.3.2.

2.2.2.4 Persistent rules in execution and planning

Every agent may have persistent rules in its beliefs that are always available, thus applications may use them in their logic (the analog of built-in functions in programming languages). The Ontonuts allows integration of such rules into the planning and execution, by providing stub annotations for them:

```
   :id :type :Ontonut
   :id :precondition {B B B}
   :id :effect {C C C}
   :id :script {empty}
```

The Ontonut stub leaves the *:script* property empty. The absence of implementation script means, that there is functionality in agent's general context that will react on appearance of statements matching the precondition and will produce statements according to the pattern of effect.

2.2.3 Distributed querying with Ontonuts

There are two main viewpoints towards distributed querying in the UBIWARE: adapter-based and service-based. The former tackles the adaptation of data sources that are external to the platform (databases, files, web services, etc.), when the latter deals with the agent-to-agent querying. Nevertheless, both target the same goal: to make distributed querying transparent to an agent (see Figure 2.5).

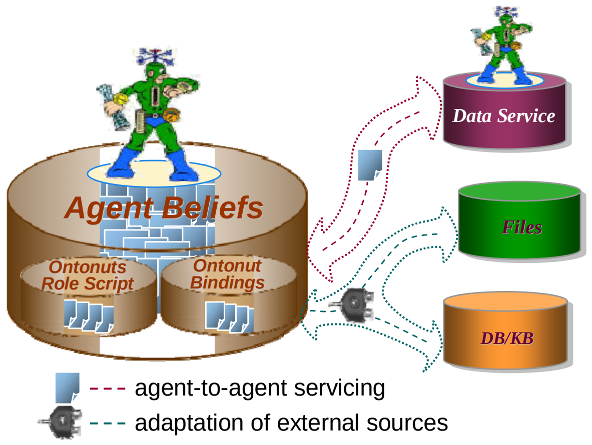


Figure 2.5 - Distributed querying in UBIWARE.

The agent-to-agent querying follows servicing paradigm and particularly data servicing discussed in (Quilitz and Leser, 2008). The adaptation of external sources (RDF-based adaptation is discussed in e.g. Langegger et al, 2007) resolves the problem of connectivity and interaction with those resources that are external to the platform, i.e. communicating with them in their native language. However, from the business logic developer's point of view, any remote resource should be transparent, in order to keep business logic script as modular and clear as possible. Ontonuts become wrapper, adapter and connector in one place.

For example, there can be a database that contains data in a native format about events of a paper machine. Let us consider the situation, when an agent needs to instantly analyze event series and, hence, perform queries regularly. An agent can have a flat solution, where all the calls to the database, as well as transformations are written within the same S-APL code that performs the analysis. Or, it may declare an Ontonut and specify a pattern that defines the query types allowed. Then the code that performs the analysis will be purely separated. And even more, if the analysis results in a certain number, it can be wrapped into the Ontonut as well, and so the end user will call the whole complex construct by putting one commitment to its beliefs, e.g.

```
{:Factory1 :hasAnalysisResult ?res}
  => {some action with ?res here}.
```

The distributed and remote querying is one of the most demanding application areas for Ontonuts, because it tackles the Enterprise Application Integration (EAI) problem as a whole. Ontonuts try to solve three main challenges: connectivity, adaptation (transformation) and integration. An illustrative example is presented on Figure 2.6.

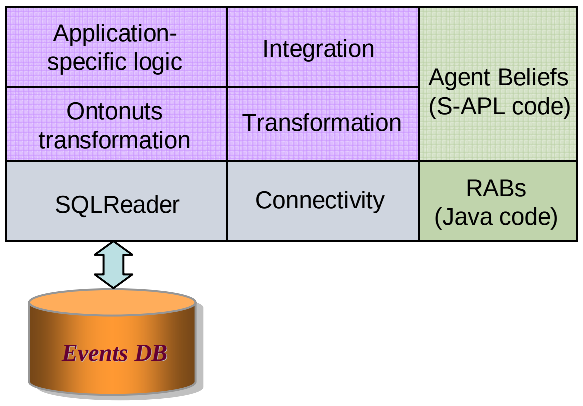


Figure 2.6 – EAI in UBIWARE.

The connectivity problem is resolved by adding new RABs, e.g. if there is a new protocol or a database provider on the market. The transformation code is a mediator between the connector’s format and the internal UBIWARE semantic model (Ontology). An Ontonut here is a semantic interface to the external database. The application-specific logic uses the Ontonut in planning to implement a transparent integration with other data and applications.

2.2.3.1 Querying use cases

Based on the scenario described in section 2.1, we have defined three typical querying cases that we use for testing and development of the Ontonuts engine:

- Event flow integration (time-based distributed query to different sources (extract events from different systems by filtering them with the same time frame)

```
{?diaryEvent :hasCommentText ?ctext.
 ?diaryEvent :hasTime ?ctime.
 ?ctime > ?timestart.
 ?ctime < ?timeend.
 ?alarmHistorian :hasAlarm ?alarm.
 ?alarm :hasTime ?atime.
 ?atime > ?timestart.
 ?atime < ?timeend.
 ?timestart = 2008.09.08T12:00.00.
 ?timeend = 2008.09.08T23:59.00. }
```

- Additional information (Get supplementary device configuration data for a particular event-based view)

```
{?diaryEvent :hasCommentText ?ctext.
 ?diaryEvent :hasTime ?ctime.
 ?ctime > 2008.09.08T12:00.00.
 ?ctime < 2008.09.08T23:59.00.
 ?diaryEvent :hasTag ?eventtag.
 ?eventtag :hasMappingTo ?node
 ?dpm :hasNode ?node.
?node :hasAlarmLimit ?alimit}
```

- Complex mining (Collect information from a set of sources, where additional computations over sub-query results are performed)

```
{?dpm :hasNode ?node.
 ?node :hasPerfIndex ?pindex.
 ?pindex < 0.5.
 ?nfiltered :filter ?node.
 ?nfiltered :hasMappingTo ?eventTag.
 ?diaryEvent :hasTag ?eventTag.
 ?diaryEvent :hasComment ?comment}
```

The first use case allows parallel execution of two independent sub-queries. The second case uses time-based filtering and sequential querying of two sources, where parameters from the first sub-query are passed as input to the second one. In the third case the sequential querying is extended with the custom filtering that may involve statistical or any other computations. Querying is a specific task for Ontonuts engine. It introduces capabilities that have a precondition for a query subgraph to meet the effect pattern of the Ontonut. Therefore we extend the engine to allow these capabilities to be used in planning and execution.

2.2.3.2 Extending Ontonuts for Distributed Querying

In a distributed querying task every Ontonut is an interface to the data source which has an associated data query pattern (effect) it replies to. The Ontonuts engine introduces an extension for data source-based Ontonuts. The extension allows the Ontonut developer not to implement all the RAB calls and S-APL transformations from the scratch, but rather define a description of the data source and transformation mappings. We call this subclass of Engine-running Ontonuts as Donuts (Database Ontonuts). The engine distinguishes the Donuts and treats them in a different way.

Naturally, the user-defined query can match several Donuts; therefore, the triggering rule invokes Action Planner. The Action Planner distinguishes sub-queries from the initial query and produces a distributed query plan. The plan is then passed to the Plan Executor. The executor handles the intermediate results of sub-queries and modifies subsequent sub-queries accordingly.

The structure of the Donuts is defined by Donuts Ontology (see Figure 2.7).

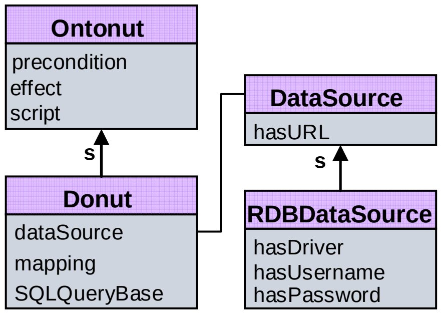


Figure 2.7 - A fragment of Donuts Ontology.

The fragment of the ontology above describes the root classes *Ontonut* and *DataSource* as well as their extensions for connectivity with relational databases (*Donut, RDBDataSource*). Similarly, other types of extensions will include type-specific facets in their descriptions. The Plan Executor uses data source descriptions for fetching the sub-queries and applies mapping definitions to transform sub-query results into the semantic form.

2.2.3.3 Mapping and adaptation in Donuts

Donuts define the interface from the agent's internal semantically structured beliefs to the external information source with different data structure. The problem of mapping the database structure to the RDF-graph is thoroughly researched in (Seaborne et al) and (Seaborne, Bizer). The authors propose an approach that maps the relational model to the RDF-model taking into account not only the structure of the database but the content as well, which is of big importance in transformation and mapping tasks. In the UBIWARE, however, we meet a slightly different task due to specifics of S-APL containerization (contextualization). We therefore try to define pattern-based interfaces that allow random structures of different complexity to be mapped, not necessarily instances of classes. In the pattern-based approach the Ontonut must fill the variable places in the pattern, and, therefore, the query to the resource (database) may have an arbitrary complexity. It is a task of a programmer to collect all the needed information and map it to the pattern values.

2.2.4 An Illustrative Example

The example presented here is based on the usage scenario described in Section 2.1 of this paper. Suppose that agent wants to extract the comment strings from the Expert's Diary database and align them with the performance indices from the Performance Monitoring database. Then, both values are printed to the command line. The commitment (query) in agent's beliefs would look like:

```
{?cres :commentID ?cid.
 ?cres :entryId ?eid.
 ?cres :commentTitle ?ctitle.
 ?cres :nodeid ?pmnode.
 ?pmnode :performanceIndex ?pindex.
}
=>
{
   {gb:I gb:do :Print} gb:configuredAs
 {x:print gb:is "The ?ctitle has
     the performance index ?pindex"}.
  }.
```

The commitment does not explicitly refer to the type of the Ontonut by the rdf:type property, which would simplify the implementation of the triggering procedure, but we apply pattern-based matching, i.e. use subtract operation over available Ontonut effect patterns and the query.

The description of the Ontonut is shown below:

```
:DiaryCommentNut rdf:type ont:Donut.
:DiaryCommentNut ont:dataSource
                     :datasourceid.

:DiaryCommentNut ont:SQLQueryBase
"SELECT CommentID, EntryID, Title
                 FROM dbo.Comment".

:DiaryCommentNut ont:mapping {
    ?commentID ont:mapsTo "CommentID".
    ?entryId ont:mapsTo "EntryID".
    ?title ont:mapsTo "Title"}.

:DiaryCommentNut :effect {
    ?rowid :commentID ?commentid.
    ?rowid :entryId ?entryid.
    ?rowid :commentTitle ?title}.
```

In the particular query, the triple that matches the identifiers:

```
?cres :nodeid ?pmnode.
```

is a bridging property between two Ontonuts, that have physical datasources behind. Therefore, it may belong to any of the Ontonuts it bridges or be an independent Ontonut at all. The way, how it is modeled is domain-specific. The property can be considered as a mapping that uses correspondence table or any other mapping algorithm. In case if the property is tied to the Ontonut, the ont:mapsTo property used in the mapping description can point to a container with the correspondence table that consists of two columns with the values to be matched.

The effect of the second Ontonut used in this example is defined as follows:

```
:PerfMonitoringResultNut :effect {
 ?pmnode :performanceIndex ?pindex.
}.
```

As soon as matching has succeeded, the matchmaking rule passes the control to the Query Planner. In this particular case, the work of Query Planner is straightforward – to decide which Ontonut to be queried first and apply query parameters for the SQL query generation. The order of execution may depend e.g. on the average expected number of records of each independent sub-query. The methods of execution planning and optimization go beyond the scope of this paper. For further reading the reader can look for e.g. (Oberheimer, Nixon, 2008).

2.2.5 Ontonuts in Agent Servicing

The notion of Ontonut introduces a capability-based presentation of the agent's action plans. Ontonuts can hide a complex action plan behind very simple precondition and effect patterns. Such capabilities can be wrapped as services, thus agent can make parts of its functionality available to the outer world by extending the Ontonut description with the service-specific attributes (i.e. QoS, contracting, etc.).

The difference between Ontonut and an agent service is their purpose of usage. Whereas Ontonut is by nature intended for internal use (simplifies agent's business logic), the service is targeted for external use (see Figure 2.8).

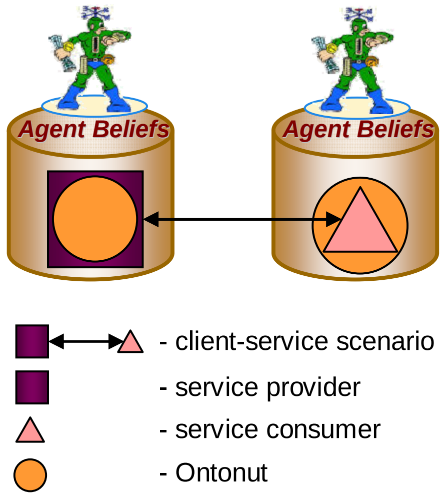


Figure 2.8 - Ontonuts vs. servicing.

As shown above, the Service Provider can wrap or use Ontonut in its service implementation. The Service Consumer can use Ontonut to ease access to a service (wrap service client, to simplify other agent's scripts).

2.5 Conclusions

The approach presented in this work targets quite specific target – to structure the S-APL code of the UBIWARE agent in order to simplify programming of the agent and allow automated goal-driven planning. Although, the work applies to quite narrow domain, it involves generic problems of agent planning and semantic programming. The strike of this paper is put on the automated planning of distributed queries and inevitably interferes with distributed RDF queries and so-called virtual graphs, when the graph being queried does not have content beneath; instead, the RDF queries are used to generate SQL queries to the corresponding database.

The approach proposed here, does not try to map models, but uses patterns to define desired result and applies queries or any other code to fill the patterns requested by the agent.

UBIWARE Deliverable D2.1:
Workpackage WP3:

3 SURPAS – Smart Ubiquitous Resource Privacy and Security

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.

According to the original project plan, the WP3's Year 2 (the *Communication* phase) had the following tasks:

Task T2.1_w3 (research): Conceptual and functional semantics, algorithms and abstract architecture: Secure Communication.

Task T2.2_w3 (development): Reference implementation of the abstract architecture in the UBIWARE prototype focusing on Secure Communication.

In other words, the plan was to work on the questions related to secure communication between agents, e.g. enabling organizational policies that do not only prohibit or allow something (like access control) but also may prescribe a certain additional actions to be taken, like encrypting every outgoing message. These questions are however already partially answered during first year, 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 2.

UBIWARE Deliverable D2.1:
Workpackage WP4:
Task T2.1_w4:
Workpackage leader: *Michal Nagy*

4 Self-Management, Configurability and Integration

UBIWARE aims to be a platform that can be applied in different application areas. This implies that the elements of the platform have to be adjustable, could be tuned or configured allowing the platform to run different business scenarios in different business environments. Such flexibility calls for existence of a sophisticated configuration layer of the platform. All building blocks of the UBIWARE platform, i.e. software agents, agent behaviors, resource adapters, etc, become subject to configuration. On the other hand, a flexible system should have a long lifespan. Hence, the platform should allow extensions, component replacements, and component adjustments during the operation time. This work package aims at introducing configurability as a pervasive characteristic of UBIWARE and developing the technology which will systemize and formalize this feature of the platform.

During WP4's Year 2 (the *System* phase), we work on issues related to configuration of the system as a whole, through distributed decision making by agents representing the system components. WP will answer the following research questions:

- How a component of a system may realize the need for re-configuration of itself of the integral system, i.e. when the previous configuration of one or more components does not seem to work anymore?
- What mechanism are needed for (re-)configuration of the integral system through local decision making of and supported by communication between agents representing components of the system (i.e. with no central decision maker)?

4.1 Self-configuration and its benefits

In the scope of the UBIWARE project the self-configuration capability is understood as a property of the system to change the nature of its elements without any external intervention of the programmer. Such system should be capable of changing its components (or component parameters) according to a common goal rather than to the explicit specification. In other words, the optimal (or nearly optimal) configuration of each agent can be discovered by the system itself.

There could be several configurations with whom a group of agents is able to perform the goal, but the question is: Which of these configurations is the best one? The best configuration could be understood as a configuration that gives us the best result for the least cost. Cost could be anything – energy, computation time, money, etc.

Sometimes finding the optimal solution can be very time consuming, especially for large domains. In these cases the goal is to find a solution that is as close as possible to the optimal one – so-called locally optimal solution. Sometimes we may be lucky and by finding the local optimum we will find also the global optimum.

In the UBIWARE platform we can identify two kinds of configuration:

- Self-configuration of the external environment
- Self-configuration of UBIWARE components

4.1.1 Self-configuration of the external environment

In this scenario a group of agents is used to configure some part of the external environment (e.g. a machine). Components of the UBIWARE platform are not being configured. They are just used to configure something that they control. This approach can be used when we want the external environment to have the property of self-configuration. Also self-manageability can be achieved, because if the system senses that there is a change in the environment, it can easily re-configure so that it will perform the task optimal (with the least cost).

4.1.2 Self-configuration of UBIWARE components

Under the term UBIWARE components we understand agents themselves. Usually agents are cooperating to accomplish one global task that is performed by all of them. In this case, the agents are trying to configure themselves in order to complete this common task.

Configuration of the platform itself could be divided into several layers. Under the term configuration we understand following:

1. Choosing the best Reusable atomic behavior (RAB) from a set of possible RABs
2. Choosing the best parameters for a RAB
3. Choosing the task or role that the software agent should play

This means that instead of hard coding we can just define a global goal together with some restrictions and the system itself will find the best configuration. This approach makes the system more scalable for future extensions. When a new element is added or removed, it may also affect the configuration of other elements of the system. However, we don't have to change the configuration of the system explicitly. Instead of that, we will just provide the new element and its neighbors with the configuration information and the system will be able to re-configure itself.

4.2 DCOP as a mathematical model

4.2.1 The model

In previous years, many researchers were dealing with the problem of multiple agents' cooperation. The mathematical model they used is called Distributed Constraint Optimization Problem (DCOP). We believe that this model can also be used for configuration purposes. DCOP assumes that we have a group of agents that cooperate in order to achieve some common goal. Each agent controls one or more variables. Each variable has exactly one domain. This domain defines all possible values that this variable could have. No other values can be chosen. A variable could be for example a role of an agent or a RAB. Under the term domain we can understand a set of possible roles or RABs. Domains don't have to be necessarily same among all agents. It means that each agent can choose values from a different domain. However these domains have to be specified. Along the variables and domains, this model defines also a set of constraints. In general they can be n-ary (between n agents). A constraint defines a possible combination of values between the agents who share the constraint. For example, some constraint may say that if agent A has chosen value X, then agent B can choose only values Y or Z, otherwise the constraint will be violated.

In the previous text we described the general DCOP model. However, this model can be simplified without the loss of its power. We can do following simplifications:

1. Each agent controls exactly one variable
    - It is well know that an agent who controls several variables can be modified to control only one variable. This can be done either by combining several variables into one or by modeling this agent as a group of smaller agents who control only one variable each.
2. Constraints are only binary
    - Each *n*-ary constraint can be decomposed into several binary constraints without loosing its meaning.

The model can be formally written as following (Maheswaran et al., 2004). Let's have a set of n agents $A_1, \dots, A_n$. As stated before, each agent controls exactly one variable – in general agent A_i controls variable v_i for $i = 1, \dots, n$. Therefore we can use the term agent and variable interchangeably. Each variable v_i can pick a value x_i from its domain D_i, formally:

$$v_i = x_i \in D_i : i = 1, \dots, n \tag{4.1}$$

Constraints can be represented using so-called constraint graph. In this graph, every node represents exactly one agent (or variable) and every edge represents exactly one constraint between the neighboring nodes (agents). To express the constraint graph more formally, we can use a symmetric matrix E. This matrix characterizes a set of edges between agents (variables) such that:

$$\begin{aligned} E_{ij} = E_{ji} = 1 &\Leftrightarrow \text{ there is a constraint between } i \text{ and } j \\ E_{ij} = E_{ji} = 0 &\Leftrightarrow \text{ there is no constraint between } i \text{ and } j \end{aligned} \tag{4.2}$$

For each pair (i, j) such that $E_{ij} = E_{ji} = 1$ let $U_{ij}(x_i, x_j)$ represent a utility when v_i has assigned value x_i and v_j has assigned value x_j. The global utility $\overline{U}(x)$ is the sum of rewards obtained from assignment *x*. An assignment *x* is a set of values given to the corresponding variables $x \in D_1 \times D_2 \times ... \times D_n$. The goal is to find an assignment $x*$ such that $\overline{U}(x*)$ will be maximal among all possible $\overline{U}(x)$. This problem can be therefore given as a group of 3 sets (X, E, U).

4.2.2 An example of a DCOP problem

The DCOP model can be easily used to describe a graph coloring problem. Let's assume that we have a graph as shown in Figure 4.1.

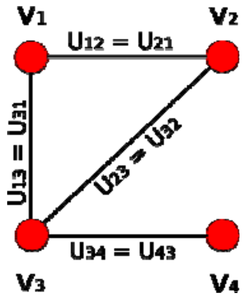
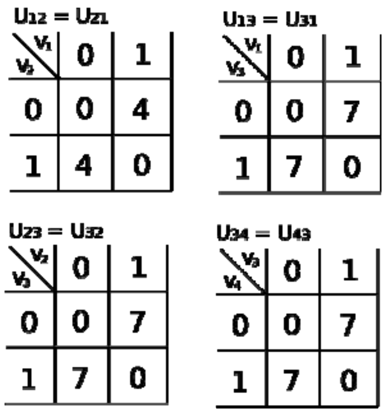


Figure 4.1 - Graph coloring problem on 4 vertex graph.

Each vertex represents one variable that stores the value of the color. Let's assume that we have only 2 possible colors (0 and 1). Thus $X_1 = X_2 = X_3 = X_4 = \{0,1\}$. In the graph there are constraints (edges) between some neighbors. The constraint says that the color on the both ends of the edge should be different. But not all constraints (edges) are equally important. Let's assume that the constraint between V_1 and V_2 is less important than others. We can model this fact by changing the utility function U_{12} as described in the Figure 2. All different combinations of colors (values) will raise the utility by 7. Only in case of U_{12} the utility is raised only by 4. If an agent should decide if it has to break the constraint $V_1 – V_2$ or other constraint, it would break constraint $V_1 – V_2$ because by breaking it, it will loose only 4 points, instead of 7 points lost by breaking any other constraint.



U12 = U21

V2 \ V1	0	1
0	0	4
1	4	0

U13 = U31

V3 \ V1	0	1
0	0	7
1	7	0

U23 = U32

V3 \ V2	0	1
0	0	7
1	7	0

U34 = U43

V4 \ V3	0	1
0	0	7
1	7	0

Figure 4.2 - Utility functions.

Now we defined all 3 required sets:

$$
\begin{aligned}
U &= \{U_{12}, U_{21}, U_{13}, U_{31}, U_{23}, U_{32}, U_{34}, U_{43}\} \\
E &= (E_{ij}) \\
X &= \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}
\end{aligned}
\tag{4.3}
$$

The knowledge and roles are distributed as following:

1. Each agent A_i can change only its variable v_i
2. Each agent A_i knows the list of his neighbors from matrix E
3. Each agent A_i knows X
4. Each agent A_i knows all U_{ij} where v_j is his neighbor

The goal is to find a color for each variable (vertex) so that the number of broken constrains is minimal in the sense of the global utility. We can see that this problem doesn't have a solution that would satisfy all the constraints. But we can find a solution that gives us a result with the highest possible utility.

There can also be slight modifications in the presented DCOP model. So far all the constraints were only soft. However, some of the constraints could be considered hard. When finding the solution, the algorithm must satisfy all hard constraints and should try to satisfy all soft constraints. By not satisfying a soft constraint, the solution can be still feasible. However by not satisfying a hard constraint, the solution is not acceptable. Hard constraints are important in so-called high-stakes scenarios where even a single broken constraint can result in a catastrophe (e.g. in a nuclear power plant).

There are several complete algorithms for solving DCOP problem (Bowring et al., 2008)(Chechetka and Sycara, 2006)(Mailler and Lesser, 2004)(Modi et al., 2005)(Petcu and Faltings, 2005)(Varakantham et al., 2006)(Yeoh et al., 2008)(Yokoo and Durfee, 1991). Some of them are globally optimal and some are only locally optimal. The next part will explain the difference between them.

4.3 Local optimization vs. global optimization

During last 20 years, several complete algorithms for solving DCOP problem were introduced. However, the DCOP problem was recognized as an NP-Hard problem (Modi et al., 2005). It means that complete algorithms provide a very high computation and communication costs when the environment is big (complex constraint graphs). Complete algorithms seeking global optimum can handle environments with about 100 agents.

Also several algorithms were introduced that were trying to find a locally optimal solution, instead of the global solution. One class of these incomplete algorithms is so called k-optimal algorithms (Maheswaran et al., 2004)(Zhang et al., 2005). In these algorithms agents dynamically form local groups to coordinate value choices. A k-optimum occurs when no group of k or fewer agents can improve the solution. In case $k = n$, the algorithm is complete.

4.4 Reconfiguration

In order to make the system self-manageable there has to be some way how to reconfigure it. As we mentioned before, a DCOP model contains a set of constraints. These constraints can depend on several attributes of the agent or the environment. Agents can sense the environment. It means that they can sense also the change of some property of the environment.

In general there are two situations when a group of agents needs to reconfigure:

1. The nature of the environment changed
2. The nature of the agent group changed

As mentioned before, an agent can sense its environment or a relevant part of it. This information about the change is stored in the belief structure of the agent. The important question is: Does this change influence the ability of the group to work efficient? If it does, then it may be necessary to reconfigure the system. If it doesn't, then there is no need to reconfigure. Therefore the next question is: How can a group of agents know that the change will affect them? The change will affect them if there is a variable that will change and the same variable is present in at least one constraint that the DCOP model contains. Since every agent can sense this variable, they know when they need to reconfigure.

The nature of the group can change if there is a new agent added to the group or an old one removed from the group. The control over these actions can be achieved by a very simple protocol that will be based on keep-alive messages together with register/unregister messages. Every time an agent is added to the group, it will register and his neighbors will start to maintain a keep-alive relationship with him. If they don't sense it any more, they will trigger the reconfiguration phase of the system. The reconfiguration phase can be triggered also when an agents unregisters. There are many working examples of similar protocols.

4.5 Comparison of existing algorithms

As stated in the previous part, there are two types of DCOP algorithms – complete and incomplete. Complete algorithms are seeking the global optimum and incomplete algorithms are seeking a local optimum as a tradeoff for a better performance. In order to measure efficiency of an algorithm, there has to be a common platform. DCOP problem, as the name states, is distributed in its nature. Therefore the evaluation of DCOP algorithms is more difficult. In (Meisels et al., 2002) a model for CCC (Concurrent Constraint Check) was presented. Using this model it is possible to evaluate the effectiveness of DCOP algorithms. It was accepted by the DCOP community and it is used to compare algorithms between each other.

4.5.1 Complete algorithms

Several requirements for a scalable complete DCOP algorithm were summarized in (Modi et al., 2005). These requirements state that the communication between agents should be local without any broadcast messages. Broadcast messages are slowing down the calculation process, because every node (agent) has to accept the message and evaluate it. Secondly, they required the algorithm to operate asynchronously. In synchronous algorithms agents have to wait for a particular message in order to continue the operation. In asynchronous algorithms this idle time is used for computation. Third requirement was to give the quality guarantee on the system performance. It means that quality/time tradeoffs are available. However not all algorithms satisfy all of these requirements.

4.5.1.1 Adopt

Adopt was presented in 2003 (Modi et al., 2003) and it is the first asynchronous algorithm for solving DCOP problem. Adopt has following attributes:

- Allows bounded-error approximation
- Asynchronous
- Each agent can change variable assignment any time it thinks it's necessary (there doesn't have to be any synchronization of actions between the agents)
- Globally optimal
- Distributed (there is no agent that acts as the central coordinator)

Before starting the Adopt algorithm, the constraints graph should be ordered into Depth-First Search (DFS) tree. There are several efficient algorithms that are used for this purpose. The memory complexity of Adopt is polynomial and the communication complexity is exponential with respect to the growing number of agents.

4.5.1.2 OptAPO

This algorithm was presented in (Mailler and Lesser, 2004). It uses the mediation technique and is asynchronous, same as Adopt. However in graph coloring domain it performs faster than Adopt. The memory complexity is polynomial and the communication complexity is exponential. However the completeness of this algorithm was challenged in (Grinshpoun et al., 2007).

4.5.1.3 DPOP

The main feature of DPOP (Distributed Pseudotree Optimization Procedure) is that it has only polynomial communication complexity. It was presented in (Petcu and Faltings, 2005). DPOP uses the idea of sum-product algorithm, but this algorithm is used only for tree structures. In this case the width of the tree is important. This parameter influences the computational speed of the algorithm. Finding a min-width tree in a graph is a NP-Hard problem. However according to the authors of the algorithm DFS traversal shows good results (Petcu and Faltings, 2005). As mentioned in the beginning, the communication complexity is polynomial. On the other hand the memory complexity is exponential. In algorithms mentioned before, the message size was the same all the time or it didn't change significantly. In the case of DPOP, the maximal message size depends on the width of the generated tree and not all messages have to have the same size. For example a problem consisting of 200 agents with 341 constraints described in (Petcu and Faltings, 2005) would result in the maximal message size of 256k values.

4.5.1.4 NCBB

NCBB (No-Commitment Branch and Bound) algorithm was presented in 2006 (Chechetka and Sycara, 2006). Similar to some of the previous algorithms, it uses a tree structure to order agents in the constraint graph. Before starting the algorithm, the graph should be ordered into DFS tree. The memory complexity is polynomial and the messages complexity is exponential. However the message size is constant (in contrast to variable size of the messages in DPOP).

The experiments show that it has better time performance than Adopt in most scenarios and similar performance to DPOP if the communication cost is close to 0.

4.5.1.5 AFB

AFB (Asynchronous Forward Bounding) algorithm was introduced in 2006 (Gershman et al., 2006). This algorithm is based on the branch and bound scheme. Agents are controlling assignments concurrently and they use a time stamp mechanism to determine the most recent assignment. The authors expect a better performance in tight problems than SyncBB algorithm. However they didn't compare it with other modern algorithms.

4.5.2 Incomplete algorithms

Incomplete algorithms are trying to find a local solution of a DCOP problem. Some of them provide a guarantee on the solution quality and some don't. In this report, we mention one group of algorithms that can provide a guarantee on solution quality. This group is called k-optimal algorithms. As the name states, k-optimal algorithms are trying to find a k-optimum. K-optimum appears when there is no group of agents smaller or equal to k that can improve the solution. In order to write it more formally, we have to define a concept of a deviating group of assignments.

A group of 2 assignments $(a,\tilde{a})$ is considered deviating if at least one agent has chosen a different value for its variable. Deviating group $D(a,\tilde{a})$ is a set of indices of agents that have chosen a different assignment for their variables, formally:

$$D(a,\tilde{a}) := \{i \in I : a_i \neq \tilde{a}_i\} \tag{4.4}$$

The distance $d(a,\tilde{a})$ is defined as the cardinality of the $D(a,\tilde{a})$ set, formally:

$$d(a,\tilde{a}) := |D(a,\tilde{a})| \tag{4.5}$$

The relative reward of an assignment a with respect to another assignment $\tilde{a}$ is given as the difference between their rewards:

$$\Delta(a,\tilde{a}) := R(a) - R(\tilde{a}) \tag{4.6}$$

An assignment a can be classified as *k*-optimal if

$$\forall\tilde{a}\ \Delta(a,\tilde{a}) > 0 \wedge d(a,\tilde{a}) \leq k \tag{4.7}$$

In other words, if we have an assignment a and we compare it with any other possible assignment that has *k* or less variables changed, we will find out that this assignment a is the best one (it gives us the biggest reward).

4.5.2.1 MGM-1

MGM-1 (Maximum Gain Message) is a very simple synchronous 1-optimal algorithm based on DBA (Distributed Breakout Algorithm) published in. MGM-1 is trying to improve the local utility in each round until there is no single agent that is capable of raising its local utility. When this state is achieved, the algorithm terminates. The result is 1-optimum.

4.5.2.2 DSA

DSA (Distributed Stochastic Algorithm) is very similar to MGM-1. It's also synchronous and it results in a 1-optimum. However, there is one small difference. In MGM-1 every round only the agent (the one with the biggest utility gain) has the right to change the value. In DSA several agents are allowed to change its variable each round. They are randomly elected, which brings some uncertainties. In some cases it may lead to unpredictable results.

4.5.2.3 MC-MGM-1

This is a modification of MGM-1. MC in this case means multiple constraints. It can also contain hard constraints. Hard constraints can't be broken. They have to be satisfied for any cost.

4.5.2.4 MGM-2 and SCA-2

Since these algorithms are 2-optimal, agents are trying to improve the overall solution in pairs. There is a mechanism based on uniform probability function that is used to create these pairs. After each pair discovers the best change to raise the utility, all agents will try to do a unilateral change as in 1-optimal algorithms. Until this step, both algorithms behave the same. However, the unilateral change in MGM-2 is done by sending gain messages between neighbors. In SCA-2 it is done by randomly choosing a number between 0 and 1. If the number is under the threshold *p*, the agent will change its value, otherwise not. Parameter *p* is known by all agents in the beginning. By choosing different values for threshold *p* you can influence the behavior of SCA-2 algorithm. In MGM-2 this number is not used.

4.5.2.5 MGM-3

With the implementation of 3-optimal algorithms new problems arise. In 2-optimal algorithms, there was one offerer and several receivers to form pairs. Offerer was sending messages to the receiver. In 3-optimal algorithms this is not possible and 2 additional cycles are need per each round to create triples. This makes MGM-3 already complicated. SCA-3 can be easily implemented just by adding uniform probabilistic function to the MGM-3 approach as in SCA-2 together with the threshold *p*.

4.6 Test cases

We provide several test cases to prove the concept of DCOP and associated algorithms. The process of transformation of a real-life problem into DCOP consists of following steps (Figure 4.3):

1. Identification of variables and domains
2. Selection of utility functions and rewards
3. Construction of the DCOP model

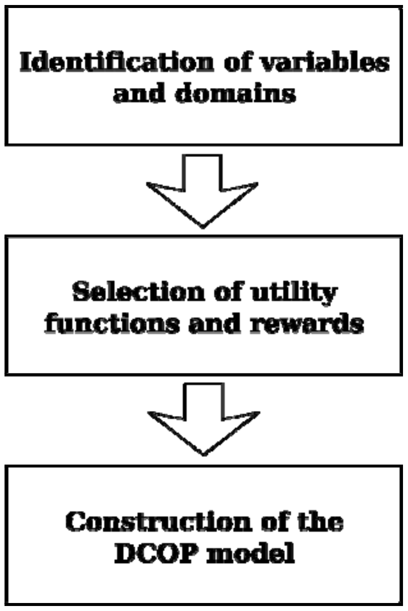

Identification of variables and domains
Selection of utility functions and rewards
Construction of the DCOP model

Figure 4.3 - The transition from a real-life problem to DCOP model.

4.6.1 Self-configuration of a WiFi network

Let's consider that we have a network consisting of several WiFi access points (antennas). These access points provide internet connectivity for one floor of a building. They were installed in a specific location and their position cannot be changed. Our goal is to adjust the power level of each WiFi access point so that the whole area is covered, but there is only a minimum of interference between the antennas. Each antenna can have 5 different power levels – 0, 10, 20, 30, 40 mW. We know how far antennas are from each other and which antennas are neighbors. The distance between the access points can be seen in Figure 4.4. Let's say that this figure also takes into consideration the wall types since they may affect the signal strength. The d_{ij} (distance *i* - *j*) values indicate the sum of power outputs of neighbors that is needed for the optimal configuration. For example, between the access point AP_1 and AP_2 the distance is $d_{12} = 5$. It means that the optimal configuration of these 2 access points would be any configuration where $x_1 + x_2 = 5$, for example $x_1 = 2$, $x_2 = 3$ or $x_1 = 4$, $x_2 = 1$.

The first step is the identification of variables and domains. In this example this is quite straightforward. There is only one variable, the power output, and it has 5 values, from 0 to 40 mW. In order to make it simpler, we can write 0, 1, 2, 3, 4 instead of 0, 10, 20, 30, 40. It doesn't change the meaning of the problem. In our case, all APs have the same power output levels. This is not a condition in the DCOP model. We could also have a heterogeneous environment, where some APs have 5 power output levels and some have only 3 or any other combination. However, to keep this example simple, we assume that all 4 of them have exactly 5 power output levels (0-4).

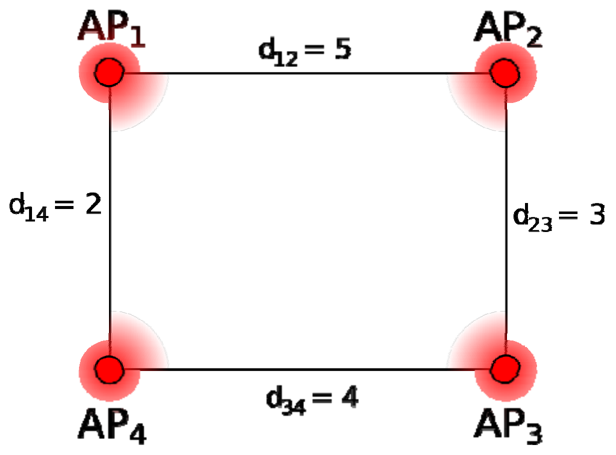


Figure 4.4 - The distance between the WiFi antennas.

The next step is to choose the utility functions u_{ij} with rewards/penalties for every pair of neighbors. Our goal is to cover the whole area, so the power output has to be high enough. But it shouldn't be too high, since then the interference could appear. Therefore we should give the biggest reward when the neighboring APs have the optimal power output (that means the sum of the neighboring power outputs is equal to d_{ij}). When the power output is lower than that value, the devices will suffer a penalty – same if the power output is too big and interference occurs. However, the penalty for not covering the area ($x_i + x_j < d_{ij}$) should be higher than the penalty for interference ($x_i + x_j > d_{ij}$). Therefore we will give penalty of -1 for every "power point" above the optimum and -3 for every "power point" under the optimum. By knowing this, the utility function u_{ij} can be defined easily as:

$$u_{ij}(x_i, x_j) = \begin{cases} [(x_i + x_j) - d_{ij}].(-1) \;\Leftrightarrow\; d_{ij} < (x_i + x_j) \\ 0 \;\Leftrightarrow\; d_{ij} = (x_i + x_j) \\ [d_{ij} - (x_i + x_j)].(-3) \;\Leftrightarrow\; d_{ij} > (x_i + x_j) \end{cases} \tag{4.8}$$

It can be seen easily that the only possible sums range from 0 (both neighbors have the lowest power output) to 8 (both neighbors have the highest power output). It means that we can use a table to express this information in a more human-readable form. We will use a table where we have the sum of neighbors' power output levels and corresponding reward/penalty (Figure 4.5).

To make the problem clearer, we construct also 4 graphs, each for one neighboring pair, where it can be easily seen how the power output influences the final reward/penalty (Figure 4.6).

The next step is to represent this problem in terms of DCOP. To accomplish this, we have to define domains set, constraint graph (matrix) and constraint functions. So far, we know the domains set and the constraint matrix, but we don't know the constraint functions. To construct them we can use tables from Figure 4.5. However we have to change them so that we cover each possible (x_i, x_j) pair. The result can be seen in Figure 4.7.

x1+x2	u12
0	-15
1	-12
2	-9
3	-6
4	-3
5	0
6	-1
7	-2
8	-3

x1+x4	u14
0	-6
1	-3
2	0
3	-1
4	-2
5	-3
6	-4
7	-5
8	-6

x2+x3	u23
0	-9
1	-6
2	-3
3	0
4	-1
5	-2
6	-3
7	-4
8	-5

x3+x4	u34
0	-12
1	-9
2	-6
3	-3
4	0
5	-1
6	-2
7	-3
8	-4

Figure 4.5 - Utility functions.

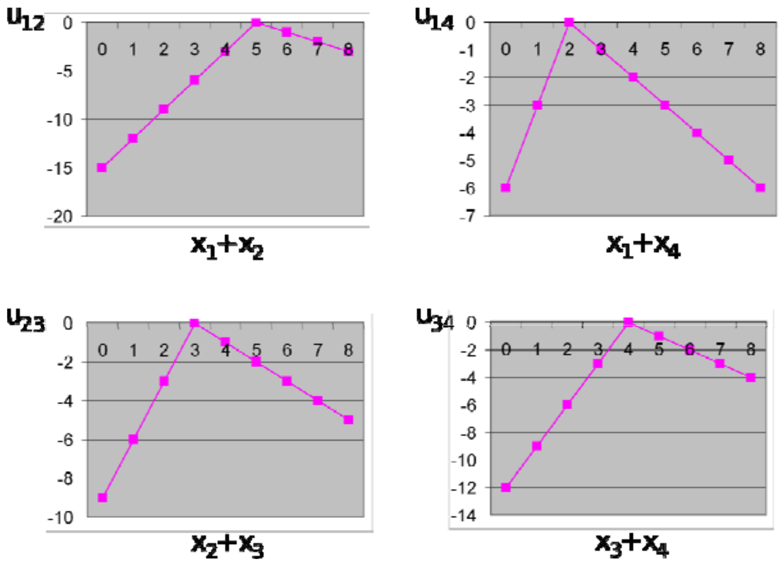


Figure 4.6 - The dependency between the power output sum and the reward/penalty.

U_{12}	0	1	2	3	4
0	-15	-12	-9	-6	-3
1	-12	-9	-6	-3	0
2	-9	-6	-3	0	-1
3	-6	-3	0	-1	-2
4	-3	0	-1	-2	-3

U_{14}	0	1	2	3	4
0	-6	-3	0	-1	-2
1	-3	0	-1	-2	-3
2	0	-1	-2	-3	-4
3	-1	-2	-3	-4	-5
4	-2	-3	-4	-5	-6

U_{23}	0	1	2	3	4
0	-9	-6	-3	0	-1
1	-6	-3	0	-1	-2
2	-3	0	-1	-2	-3
3	0	-1	-2	-3	-4
4	-1	-2	-3	-4	-5

U_{34}	0	1	2	3	4
0	-12	-9	-6	-3	0
1	-9	-6	-3	0	-1
2	-6	-3	0	-1	-2
3	-3	0	-1	-2	-3
4	0	-1	-2	-3	-4

Figure 4.7 - Utility functions.

Now, we know all 3 elements, so we can construct the DCOP model as following:

$$\begin{aligned} E &= \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \\ X &= \{0,1,2,3,4\} \times \{0,1,2,3,4\} \times \{0,1,2,3,4\} \times \{0,1,2,3,4\} \\ U &= \{U_{12}, U_{14}, U_{23}, U_{34}\} \end{aligned} \tag{4.9}$$

After we created the DCOP model, we can distribute the information among the agents. Then we need to initialize the variables. In this case we can initialize them all to 0 (no power output). The initial configuration of the system will be $x_0 = (0,0,0,0)$. We can easily see that this configuration gives us the following reward:

$$\begin{aligned} R(x_0) &= u_{12}(x_1, x_2) + u_{14}(x_1, x_4) + u_{23}(x_2, x_3) + u_{34}(x_3, x_4) = \\ &= u_{12}(0,0) + u_{14}(0,0) + u_{23}(0,0) + u_{34}(0,0) \\ &= (-15) + (-6) + (-9) + (-12) \\ &= -42 \end{aligned} \tag{4.10}$$

As we can see this is the worst reward possible, so there is a lot of space for improvement. Now we can start one of algorithms mentioned in the previous section. If it's a complete algorithm sooner or later it will come to the global optimum and if it's a k-optimal algorithm, it will come to a k-optimum. For example in this case 1-optimum could be configuration $(2,3,0,4)$ with the global utility $R(2,3,0,4) = -4$.

4.6.2 Best role configuration

The second example is devoted to the problem of the best role configuration. There is a group of 4 agents that can communicate together. The task is to choose a pair of agents – the fist one will play *collecting* role and the second one will play *storing* role. The rest of the agents won't play any role.

There are several requirements put on the multi-agent system:

1. There can be exactly one agent playing *collecting* role
2. There can be exactly one agent playing *storing* role
3. Agent playing *storing* role must have database access
4. Agent playing *collecting* role must have HTTP access
5. Agent playing *storing* role must not have HTTP access

There are also different communication links between agents. Some are fast, some are slow. The speed s_{ij} of each link (AP_i, AP_j) is shown in Figure 4.8. Higher number means higher speed. Speed 0 means that this pair cannot communicate at all.

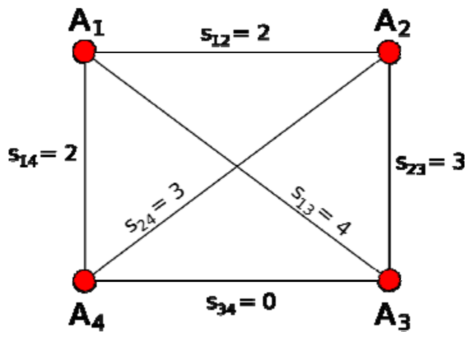


Figure 4.8 - Constraints graph.

The capabilities of each agent are shown in Table 4.1.

Table 4.1 - Agents' capabilities.

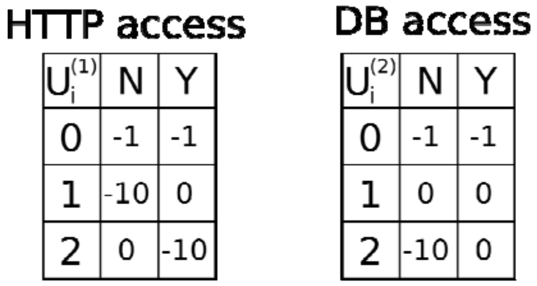
Agent	HTTP access	Database access
A_1	Y	N
A_2	Y	N
A_3	N	Y
A_4	Y	Y

The first step is the identification of variables and domains. In this case we have only 1 variable per agent – the role. Each variable can have 3 possible values indicating the role that the agent will play. The roles and corresponding numbers can be seen in Table 4.2.

Table 4.2 - Number-role mapping.

Number	Role name
0	no role
1	Collecting
2	Storing

The next step is to choose utility functions. We have to "decode" all the conditions using utility functions. Therefore we will have 2 different types of utility functions – unary and binary. Using unary utility functions we can express the need to have HTTP access in the case of *collecting* role and the need to have database access in the case of *storing* role. We can express it as in Figure 4.9. The *Y* column indicates the reward/penalty if the agent has the discussed feature. Column *N* gives the reward/penalty in case the agent doesn't have this feature.


HTTP access

$U_i^{(1)}$	N	Y
0	-1	-1
1	-10	0
2	0	-10

DB access

$U_i^{(2)}$	N	Y
0	-1	-1
1	0	0
2	-10	0

Figure 4.9 - Unary utility functions.

For every broken condition there is a reward -10. In this case it is de facto penalty of 10, because the reward is a negative number. There is also a reward of -1 when the agent isn't playing any role. The reason for it is that we should motivate the agent to change its initial state and pick a role. However, the requirements must be met. If the agent picks a role for which it doesn't meet the requirements, it will receive a reward -10. In this case, the agent will pick no role, because that would give him a reward -1, which is better than -10.
Now we need to create a second group of utility functions – binary utility functions. They will be created from the constraints graph. The result can be seen in Figure 4.10. If two neighboring nodes picked different roles, they would receive a reward equal to the speed of the link between them (red color in the figure). If they picked the same role, they would get a reward of -10 (blue color in the figure). By doing this we assure that there will be only one *collecting* role and one *storing* role. Having two same roles is just very expensive (too big penalty). If one of the agents doesn't play any role (the variable that it controls is equal to zero), there will be no reward/penalty (black color in the figure).

U_{ij}	0	1	2
0	0	0	0
1	0	-10	s_{ij}
2	0	s_{ij}	-10

Figure 4.10 - General binary utility function.

Now, we have all necessary information to complete the whole DCOP model. The constraint matrix can be easily constructed from the constraints graph in Figure 4.8. The rest was discussed earlier. Formally the model is:

$$E = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$X = \{0,1,2\} \times \{0,1,2\} \times \{0,1,2\} \times \{0,1,2\} \tag{4.11}$$

$$U = \{U_{12}, U_{23}, U_{34}, U_{14}, U_{13}, U_{24}, U_1^{(1)}, U_1^{(2)}, U_2^{(1)}, U_2^{(2)}, U_3^{(1)}, U_3^{(2)}, U_4^{(1)}, U_4^{(2)}\}$$

When we have the description of the problem in the form of a DCOP model, we can run an algorithm to solve it. If we ran a complete algorithm, the result would be the configuration $x^* = (1,0,2,0)$. This is the best possible configuration that gives us the global reward of 4. The result can be interpreted as following:

1. Agent A_1 will play *collecting* role
2. Agent A_3 will play *storing* role

4.7 Conclusion and future work

In this part of the report we presented a solution to the self-configuration and self-manageability problem. We understand the term self-configuration as the ability of the multi-agent system to change the nature of its elements without any external intervention of the programmer. Instead the direct intervention, the system is able to configure itself based on a set of rules.

As a mathematical model we have chosen the DCOP model. The DCOP model contains reward/penalty functions that govern the process of the self-configuration. The goal is to find a configuration that has the biggest global utility (reward) based on the utility functions.
We also explained the difference between algorithms seeking the global optimum and algorithms seeking a local optimum. From the group of algorithm seeking a local optimum, we described so-called k-optimal algorithms and explained what a k-optimum means. The report also contains a comparison of several complete and incomplete algorithms.

The last part of the report was devoted to 2 examples where we showed step-by-step how a real-life problem can be converted into a DCOP problem and later solved using the UBIWARE platform. The first example showed how the DCOP model can be used to configure a group of WiFi access points in order to work efficiently. This example contained relatively simple binary constraints. The second example was dealing with the problem of best role selection. This problem was more complicated and it contained both unary and binary constraints.

In the future we will concentrate on the implementation of the DCOP model into the UBIWARE platform together with several complete and incomplete algorithms. Then we will run several benchmark tests on various problem domains to find the best algorithm for solving the particular problem. We assume that not all algorithms will be suitable for all classes of problems. Therefore we think that it is important to implement several algorithms and let the developer choose the one that fits the best.

UBIWARE Deliverable D2.1:
Workpackage WP5:
Task T2.1_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.

WP5's Year 2 elaborates on probably the most important part of 4i vision, which can be called "context provision". Especially when considering a human, presenting information on a resource of interest alone is not sufficient - information on some "neighboring" objects should be included as well, which form the *context* of the resource. For example, a resource can be presented on a map thus shown in the context of objects which are spatial (geographic) neighbors of it. What is important is that in different decision-making situations, different contexts are relevant: depending on the situation the relevant neighborhood function may be e.g. physical spatial, data-flow connectivity, what-affects-what, similar-type, etc. The ability to determine what type of context in right one for the situation and collecting the information that forms the context of that type for a specific resource is central in 4i vision.

During WP5's Year 2 (the *Context-awareness* phase), therefore, the following research questions are to be answered:

- What should be the architecture of MetaProvider-services so that they will be able to effectively retrieve, integrate and deliver the context information both to for presenting to humans (in a visual form) and for agents' processing (semantic data)?
- What should be the architecture of the Intelligent GUI Shell, so that it will allow situation-dependent selection of MetaProviders (i.e. different types of context) and cross-MetaProvider browsing and integration?

5.1 SmartInterface: GUI-Shell enhancements

With a purpose to make interaction with a SmartInterface more user-friendly and do not demand a lot of manipulation from user, we defined default visualization contexts for resource classes and MetaProviders for certain visualization context. To complete this enhancement, we add personalization to the prototype and user will be able to choose favorite visualization module (MetaProvider) for certain resource in certain visualization context.

Another enhancement concerns a smart and intelligent technique for automatic dynamic selection of a visualization context. The logic is based on a history of visualization contexts and resources that users have browsed/visualized previously. This context ranking technique allows us to sort a list of visualization contexts in more appropriate order for user and give him/her a hint for next logical step in though resource browsing process. Thus, it can become a smart search system that leads the user in proper direction/way.

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

5.1.1 Context ranking function

Current implementation of 4i(FOREYE) Browser provides just simple selection of visualization contexts from the list of existing (see Figure 5.1). To make the browsing process more user-friendly and reduce amount of useless manipulations from the user side, browser automatically visualized currently selected resource in a context that is considered as a default context for the class of selected resource. Such feature makes browser more dynamic and handy for the user, but can be useless sometimes and imposes unnecessary default visualization on user. Keeping going to the direction of context selection enhancement, we decided to enhance the browser with context ranking functionality. This add-on will rank (sort by relevance to the current situation) the list of visualization contexts for user depending on the user browsing history (current browsing route - a sequence of visualized resources and correspondent visualization contexts) and the experience of other users (history of browsing routes).

The idea is to keep all the user browsing routes in database and compare browsing route of the current user with them. So, the more similar the current route to some routes from the history database, the higher probability that a visualization context (chosen by predecessors) for current resource fits the needs of current user. The result of such add-on is a list of sorted visualization contexts in a context of browsing history and experience of predecessors (see

Figure 5.2). The most relevant context should be applied for visualization of recently chosen resource. If there is no matching with the previous browsing routes in the history database, then the default context will be chosen (as has been implemented in the previous version of the browser).

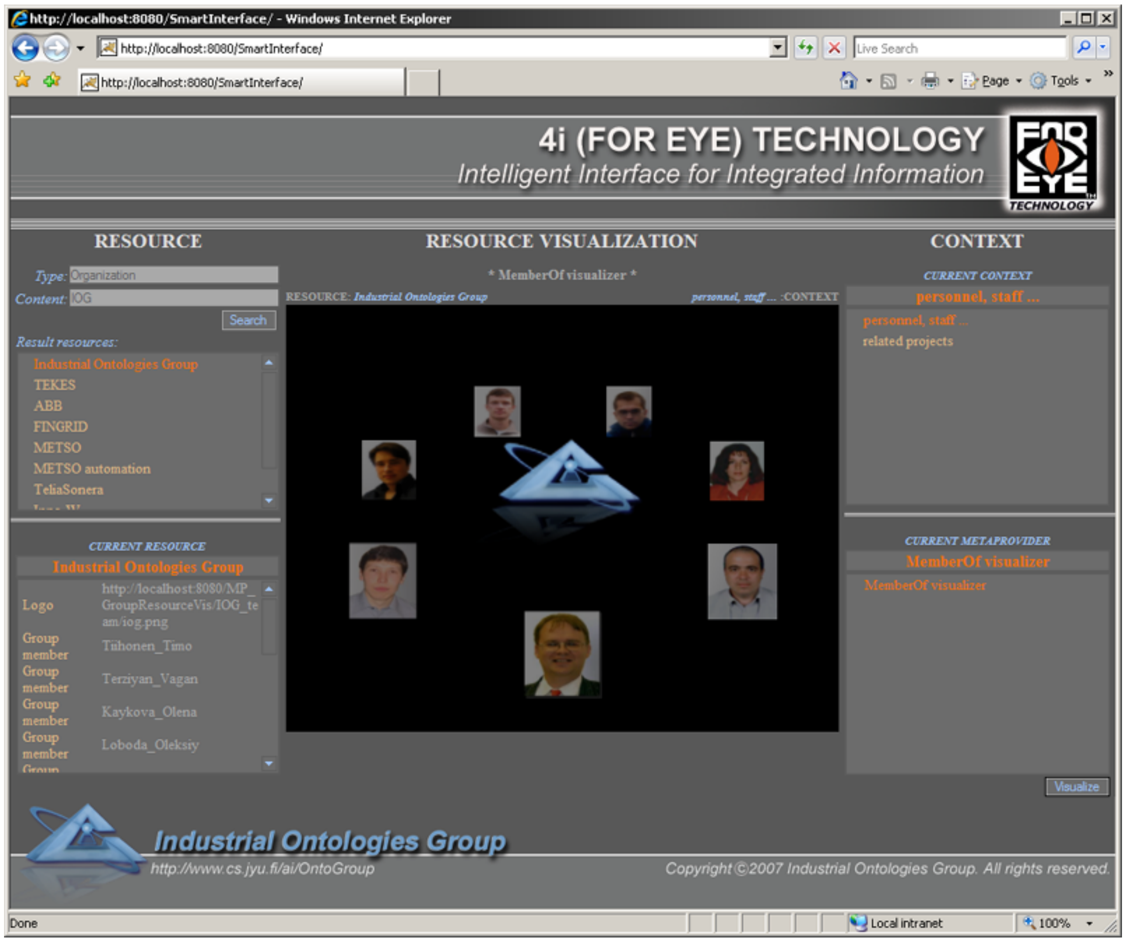

http://localhost:8080/SmartInterface/ - Windows Internet Explorer
4i (FOR EYE) TECHNOLOGY
Intelligent Interface for Integrated Information
RESOURCE
RESOURCE VISUALIZATION
CONTEXT
Type: Organization
Content: IOG
Search
Result resources:
Industrial Ontologies Group
TEKES
ABB
FINGRID
METSO
METSO automation
TeliaSonera
* MemberOf visualizer *
RESOURCE: Industrial Ontologies Group
personnel, staff ... :CONTEXT
CURRENT CONTEXT
personnel, staff ...
related projects
CURRENT RESOURCE
Industrial Ontologies Group
Logo
http://localhost:8080/MP_GroupResourceVis/IOG_team/iog.png
Group member
Tiihonen_Timo
Terziyan_Vagan
Kaykova_Olena
Loboda_Oleksiy
CURRENT METAPROVIDER
MemberOf visualizer
Visualize
Industrial Ontologies Group
http://www.cs.jyu.fi/ai/OntoGroup
Copyright ©2007 Industrial Ontologies Group. All rights reserved.
Done
Local intranet
100%

Figure 5.1 - Current version of 4i(FOR EYE) browser.

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

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

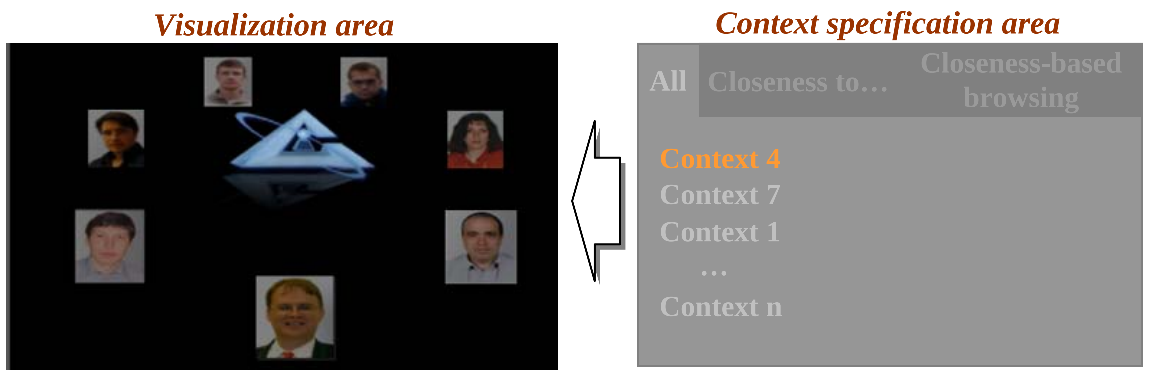


Figure 5.2 – Context-based resource visualization.

5.1.2 Similarity/Closeness visualization context

Sometimes user faces a need to find similar/close resources to the existing (selected) one. Thus, visualization of the resources in a context of their similarity/closeness becomes inherent functionality of the 4i(FOR EYE) Browser (see Figure 5.3). We single out two types:

- **closeness** closeness to resources of another different class;
- **similarity** closeness to resources of a same class (can be considered as a particular case of *closeness* type);

Calculation of a distance measure is based on a set of preselected relevant/common properties and a measure of significance those properties on a distance measure. Each pair of resource classes has own set of relevant properties. But still, this attributes are configurable. User may reduce amount of properties (select just subset of them) to be considered in matchmaking algorithm and adjust/customize significance of the properties. If user does not provide detailed specification of the context, then matchmaking bases on all common contextual properties. As a result of a matchmaking algorithm, we have a matrix of distances between subject resource and other resources that helps us clearly show closeness via visual representation in GUI.

5.1.3 Closeness-based resource browsing

Approach of finding similar/close resources can be applied for closeness-based resource browsing (see Figure 5.4). The idea is to visualize the resource in a context of its closeness to other resources based on just only one property selected by user. Then user can change the focus and select another resource to visualize it in a context of its closeness to other resources based on any its property. Such closeness can be searched among all relevant resources or among specific class of resources specified by user.

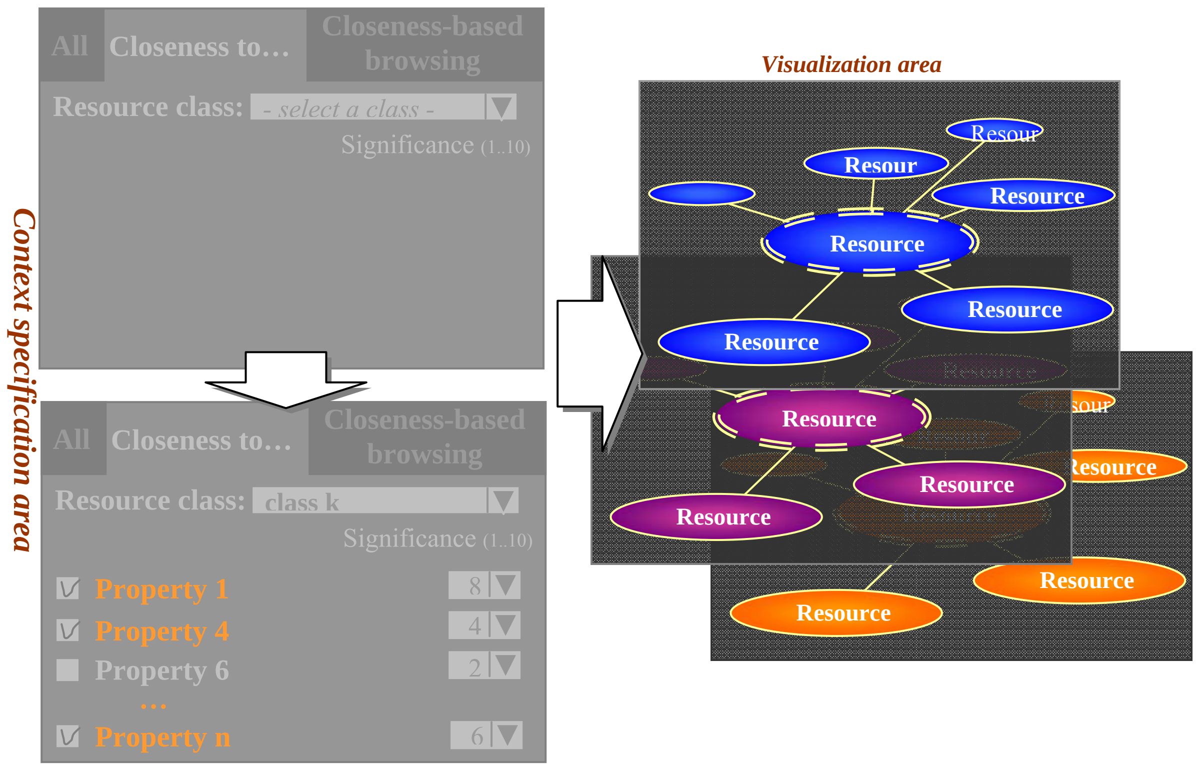


Figure 5.3 – Resource visualization in a context of its closeness to other resources.

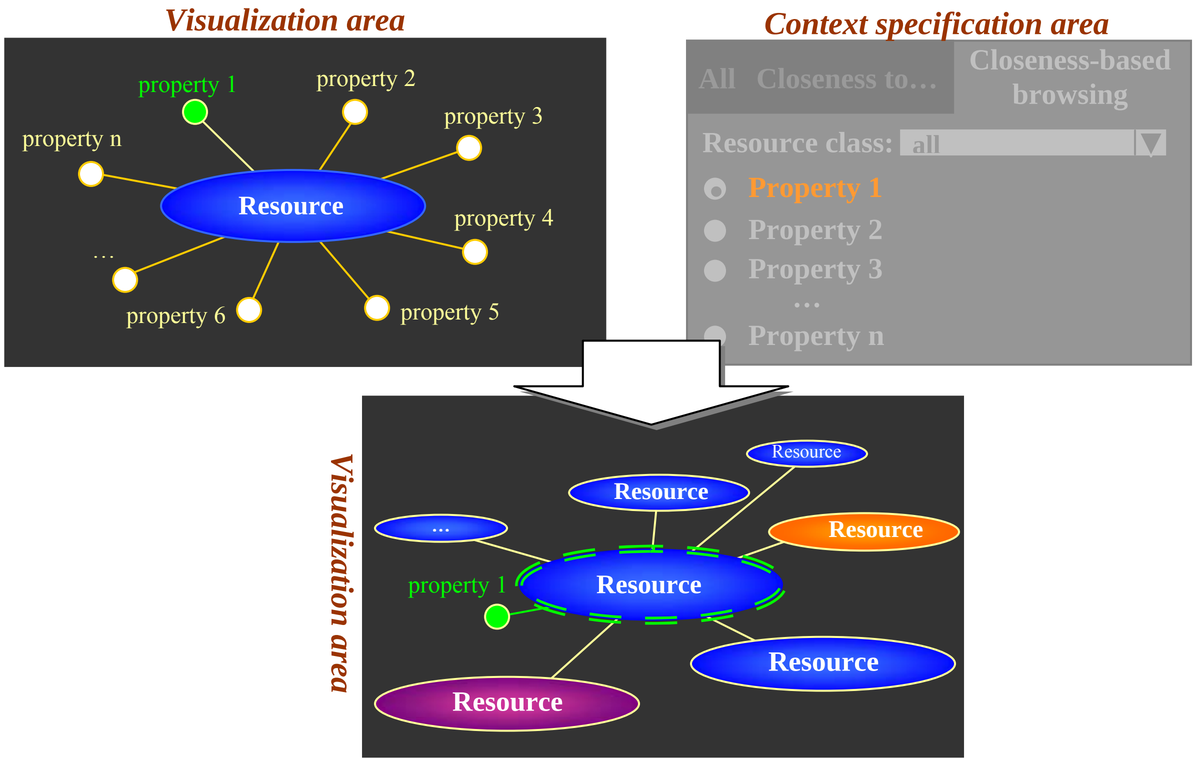


Figure 5.4 – Closeness-based resource browsing.

5.2 Distance measurement function

As the most used property value types we considering: *numerical* (numbers, dates) and *nominal* (textual strings (either defined on a finite or infinite set of possible values)) attributes. Different calculation procedures will be applied for different value types. Distance for numerical attributes can be calculated based on more or less well-known metrics; however, distance for nominal attributes (unstructured textual strings) is a challenge. For the automated distance measuring, row text fields should be additionally supported by list of strings (keywords/key-sentences). Distance measuring algorithm for the lists of strings can be invented and designed based on utilization of Google, WordNet, Wikipedia or other available online Web-services.

- **Number:** Distance measuring bases on normalization all of the values from whole sample of them. The formula of a distance between two values v_i and v_k is:

$$d(v_i, v_k) = \left| \frac{v_i - v_k}{v_{\max} - v_{\min}} \right|, \tag{5.1}$$

  where $v_{\max}$ and $v_{\min}$ are the maximum and minimum values from the sample.

- **Interval:** The problem of ordering intervals (crisp and fuzzy) is of perennial interest because of its direct relevance to the practical modelling and the optimization of real world processes. Theoretically, intervals can only be partially ordered and hence cannot be compared in the usual sense. However, when intervals are used in practical applications or when a decision has to be made among alternatives, the comparison of intervals becomes necessary. Comparison of dates is quite challenging task. If we consider just single points on a time line, we can apply distance measurement techniques used for numbers. But, if we have a deal with time intervals, then it becomes more complex and not trivial. There are different relations between intervals and we have to take them into account (see Figure 5.5).

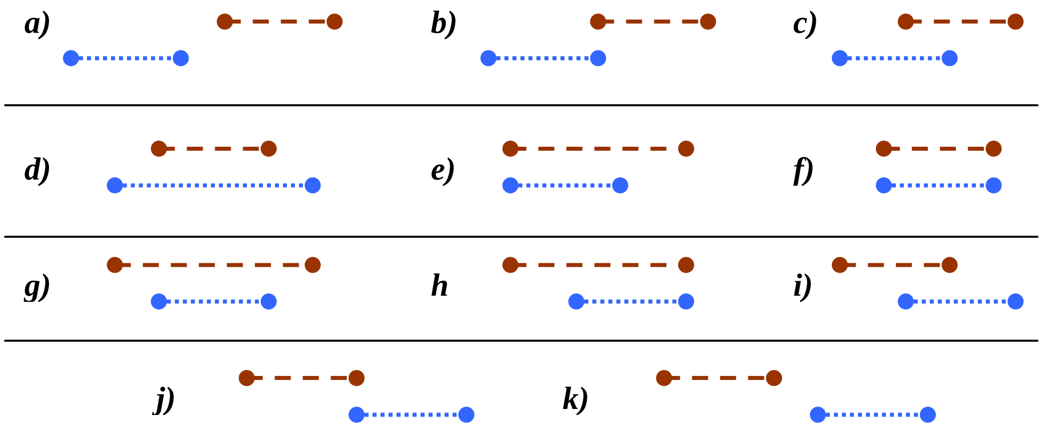


Figure 5.5 – Relations between intervals.

For the first step, we have decided to focus on main aspects of time periods comparison: durations of the periods and distance between the intervals. And the simplest formula that implicitly take into account these parameters (see Figure 5.6) is:

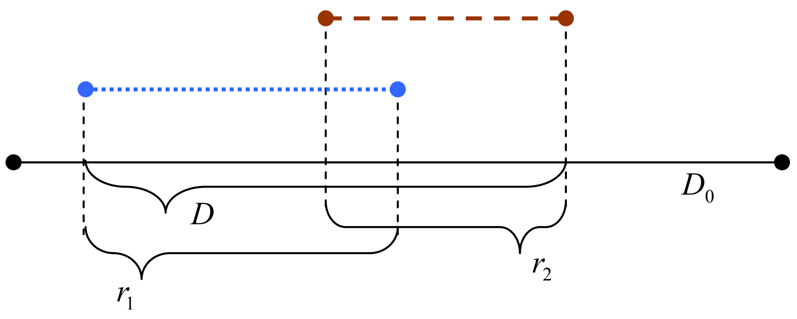


Figure 5.6 – The intervals comparison (Type1).

$$d([a_i, b_i], [a_j, b_j]) = \frac{D - r}{D_0}, \tag{5.2}$$

where

$$r = \frac{r_1 + r_2}{2} = \frac{(b_1 - a_1) + (b_2 - a_2)}{2},$$

$$D = \max(b_i, b_j) - \min(a_i, a_j),$$

$$D_0 = \max_{\forall p}(b_p) - \min_{\forall q}(a_q).$$

If we are going to explicitly control the influence of interval attributes and to tune their significances, then we can use another more complex formula. For such intervals distance measurement we are going to use formula that takes into account the distance between the intervals' centers and difference between the lengths of the intervals (see Figure 5.7):

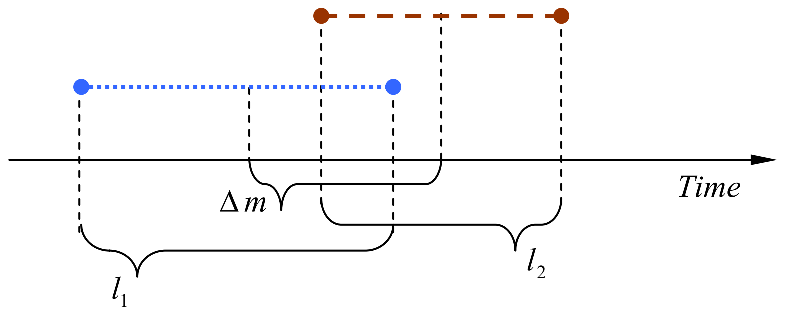


Figure 5.7 – The intervals comparison (Type2).

$$D(t_i, t_k) = \sqrt{k_m \cdot \left(\frac{|m_i - m_j|}{M}\right)^2 + k_l \cdot \left(\frac{|l_i - l_j|}{l_{\max}}\right)^2}, \tag{5.3}$$

$$D(t_i, t_k) = \sqrt{k_m \cdot \left(\frac{\Delta m}{M}\right)^2 + k_l \cdot \left(\frac{\Delta l}{l_{\max}}\right)^2}, \tag{5.4}$$

where m_i and m_k are the values of the intervals' centers on a time line, l_i and l_k are the durations/lengths of the intervals, M and $l_{\max}$ are maximum distance between the centers of two intervals and maximum duration/length of interval from a sample set. The coefficients k_m and k_l regulate significance of the distance between the intervals and difference between the lengths of the intervals. The condition for those coefficients is:

$$k_m + k_l = 1, \tag{5.5}$$

But still, depending on sample, $\frac{\Delta m}{M}$ and $\frac{\Delta l}{l_{\max}}$ can differently influence on a result, even if the coefficients will be equal. We have to modify the function with a correction part that balances centers of masses.

$$D(t_i, t_k) = \sqrt{k_m \cdot \left(0.5 + \frac{0.5 \cdot \left(\frac{\Delta m}{M} - m'\right)}{\max\limits_j \left|\frac{\Delta m}{M}_j - m'\right|}\right)^2 + k_l \cdot \left(0.5 + \frac{0.5 \cdot \left(\frac{\Delta l}{l_{\max}} - l'\right)}{\max\limits_r \left|\frac{\Delta l}{l_{\max}}_r - l'\right|}\right)^2}, \tag{5.6}$$

where m' and l' are medians of corresponding samples.

Now we can compare the results of both approaches and chose an appropriate one depending on a goal. Let us consider a sample of intervals (see Figure 5.8) and find the closeness of the intervals ($i_1 .. i_{10}$) to the initial one (i_0).

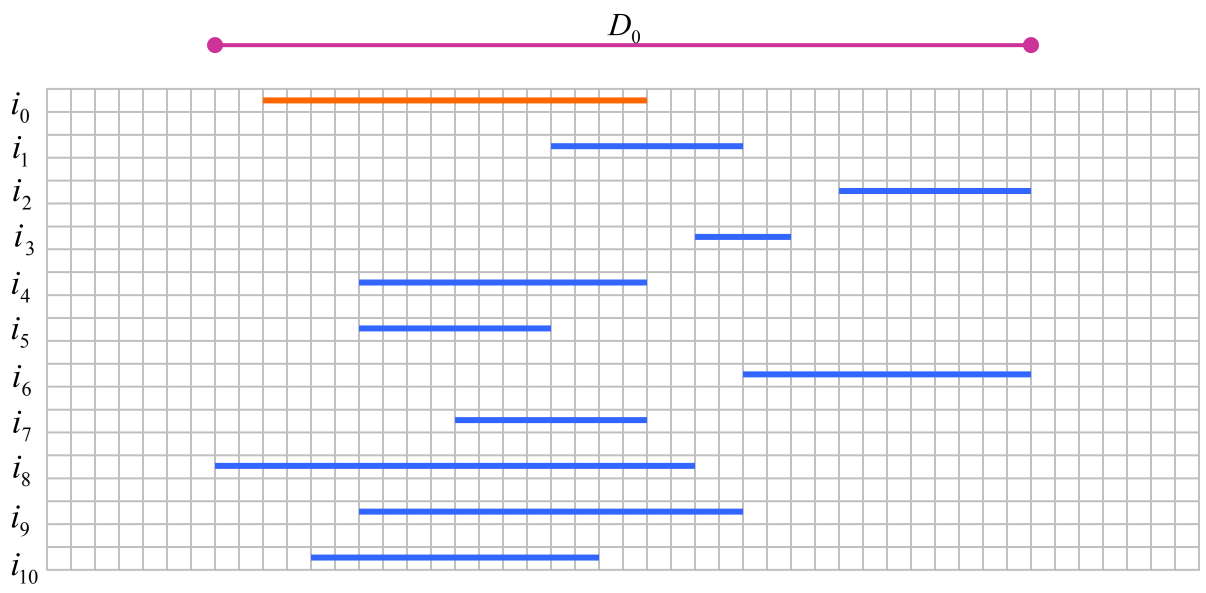


Figure 5.8 – Sample of the intervals.

Table 5.1 – Comparison of interval distance measurement approaches (*median*).

Interval	Measurement Type1		Measurement Type2		Measurement Type2		Measurement Type2	
	Distance to i_0	range	Distance to i_0, $k_m = 0.5$ $k_l = 0.5$	range	Distance to i_0, $k_m = 0.7$ $k_l = 0.3$	range	Distance to i_0, $k_m = 0.3$ $k_l = 0.7$	range
i_1	0,2353	*3*	0,6461	*6*	0,4067	*6*	0,4282	*7*
i_2	0,5882	*6*	0,8498	*8*	0,8333	*9*	0,611	*8*
i_3	0,3529	*4*	0,8839	*9*	0,6938	*8*	0,8688	*9*
i_4	0,0588	*1*	0,3888	*3*	0,1673	*2*	0,1352	*3*
i_5	0,1176	*2*	0,5408	*4*	0,2317	*4*	0,3532	*5*
i_6	0,5294	*5*	0,7036	*7*	0,6485	*7*	0,3415	*4*
i_7	0,1176	*2*	0,5892	*5*	0,3083	*5*	0,3860	*6*
i_8	0,0588	*1*	0,1258	*2*	0,1317	*1*	0,12	*2*
i_9	0,1176	*2*	0,125	*1*	0,175	*3*	0,075	*1*
i_{10}	0,0588	*1*	0,1258	*2*	0,1317	*1*	0,12	*2*

Let us consider m' and l' as arithmetic average values instead of medians of corresponding samples in interval comparison formula (Type2). Table 2 shows as corresponding results and interval ranges.

Table 5.2 – Comparison of interval distance measurement approaches (*arithmetic average*).

Interval	Measurement Type1		Measurement Type2		Measurement Type2		Measurement Type2	
	Distance to i_0	range	Distance to i_0, $k_m = 0.5$ $k_l = 0.5$	range	Distance to i_0, $k_m = 0.7$ $k_l = 0.3$	range	Distance to i_0, $k_m = 0.3$ $k_l = 0.7$	range
i_1	0,2353	3	0,6091	6	0,5845	6	0,6327	7
i_2	0,5882	6	0,9218	9	0,9129	9	0,7817	8
i_3	0,3529	4	0,8619	8	0,8	8	0,9196	9
i_4	0,0588	1	0,3257	3	0,3228	2	0,3289	3
i_5	0,1176	2	0,5016	4	0,4176	4	0,5732	4
i_6	0,5294	5	0,6947	7	0,7945	7	0,5779	5
i_7	0,1176	2	0,5475	5	0,4918	5	0,5979	6
i_8	0,0588	1	0,2914	2	0,2728	1	0,3089	2
i_9	0,1176	2	0,2786	1	0,3295	3	0,2156	1
i_{10}	0,0588	1	0,2914	2	0,2728	1	0,3089	2

o **Strings** The progress of information technology has made it possible to store and access large amounts of data. However, since people think in different ways and use different terminologies to store information, it becomes hard to search each other's data stores. With the advent of the Internet, which has enabled the integrated access of an ever-increasing number of such data stores, the problem becomes even more serious.

Well-known PEBLS distance evaluation for nominal values can be applied for nominal attributes that have values defined on a finite set of possible values.

Text field type 1: Let's consider a text field that extended/enriched by set of sub-fields/attributes with values defined on a finite set of possible values. The distance d_j^l between two values v_1 and v_2 for j attribute in a context of attribute l is:

$$d_j^{l\,2}(v_1, v_2) = \sum_{i=1}^{k} \left(\frac{C_{1i}}{C_1} - \frac{C_{2i}}{C_2} \right)^2, \qquad (5.7)$$

where C_1 and C_2 are the numbers of instances in the training set with selected values v_1 and v_2, C_{1i} and C_{2i} are the numbers of instances from the i-th class, where the values v_1 and v_2 were selected, and k is the number of the classes of instances (from values of attribute l).

Thus, general distance d_j between two values v_1 and v_2 for j attribute is:

$$d_j(v_1, v_2) = \frac{\sum_{l=1, l \neq j}^{n} d_j^l(v_1, v_2)}{n-1}, \tag{5.8}$$

where n is a number of attributes (sub-fields).

Finally, distance between two objects based on such complex field can be calculated based on following formula:

$$D(V_1, V_2) = \sqrt{\sum_{\forall i, v_1^i \in V_1, v_2^i \in V_2} \omega_i \cdot d_i(v_1^i, v_2^i)^2}, \tag{5.9}$$

where $d_i(v_1^i, v_2^i)$ is a distance by certain attribute (sub-field) and ω_i is weight for attributes. The requirement for the weights is:

$$\sum_{i=1}^{n} \omega_i = 1, \tag{5.10}$$

where n is a number of attributes.

If there is no differences in significance of the attributes, then $\omega_i = 1/n$ and

$$D(V_1, V_2) = \sqrt{\frac{\sum_{\forall i, v_1^i \in V_1, v_2^i \in V_2} d_i(v_1^i, v_2^i)^2}{n}}. \tag{5.11}$$

If some attributes (sub-fields) are not specified, it means that there is no information in the current text field that concerns those attributes. The distances between values of such attributes are - “0”.

Text field type 2: Let’s consider a text field that extended/enriched by list of values defined on a finite set of possible values (let’s say a list of key-strings). Such lists can contain different number of instances. The distance between two objects based on such a field can be calculated based on following formula:

$$D(X, Y) = \frac{N'}{N}, \tag{5.12}$$

where N' is a number of matched/equal instances and N is a general number of all instances in the lists of two comparable objects.

The Semantic Web aims to use semantics in the retrieval process, where the semantics is captured in ontologies or at the very least in concept hierarchies. The task then is to find pairs of concepts from different meta-data schemas that have an equivalent meaning, a problem known as ontology matching. This problem has been extensively studied in the Semantic Web and elsewhere, see (Shvaiko and Euzenat, 2005)(Giunchiglia and Shvaiko, 2004)(Shvaiko, 2004)(de Bruijn, 2004)(Rahm, 2001) for recent survey papers. However, in many realistic domains, it is impossible to give precise concept definitions, and consequently no crisp notion of concept equivalence exists. Ontology matching must then be redefined to finding a concept with the closest meaning in the other schema when an equivalent one does not exist. We then require mechanisms that are able to find approximate correspondences rather than exact ones.

To find the distance between two terms, we can utilize a dissimilarity measure, called Normalized Google Distance (NGD), introduced in (Cilibrasi, 2007). NGD takes advantage of the number of hits returned by Google to compute the semantic distance between concepts. The concepts are represented with their labels which are fed to the Google search engine as search terms. Given two search terms x and y, the normalized Google distance between x and y, $NGD(x,y)$, can be obtained as follows

$$NGD(x,y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log M - \min\{\log f(x), \log f(y)\}}, \qquad (5.13)$$

where $f(x)$ is the number of Google hits for the search term x, $f(y)$ is the number of Google hits for the search term y, $f(x,y)$ is the number of Google hits for the topple of search terms x y and M is the number of web pages indexed by Google[1].

Intuitively, NGD(x, y) is a measure for the symmetric conditional probability of co-occurrence of the terms x and y: given a web-page containing one of the terms x or y, $NGD(x,y)$ measures the probability of that web-page also containing the other term.

If we are dealing with nominal attributes that have values defined on an infinite set of values, it is also reasonable to utilize existing remote services that measure Semantic Relatedness of the strings. One of those services is Measures of Semantic Relatedness 2 (MSRs). MSRs are computational means for calculating the association strength between terms. More specifically, MSRs take the form of computer programs that can extract relatedness between any two terms based on large text corpora. MSRs have been used to produce models of human web-browsing behavior (Pirolli, 2005), augmented search engine technology (Dumais, 2003), semantic relevancy maps (Veksler and Gray, 2007), essay-grading algorithms for ETS (Landauer et al., 1998), and could be useful for any cognitive models or AI agents that have to deal with text.

Lack of Standardization in MSR Services: Although there are multiple MSR services that are readily available to the research community, these services are (1) scattered and (2) inconsistently formatted. All of the available MSR web servers use different input/output standards, making it less than ideal for

[1] Currently, the Google search engine indexes approximately ten billion pages ($M \sim 10^{10}$).
[2] Measures of Semantic Relatedness (MSRs) - http://cwl-projects.cogsci.rpi.edu/msr/msr-about.html

researchers that may want to compare, contrast, alter, and average these measures. Some MSRs are available to download, but these technologies are even more diverse in protocol, and are much harder to use. To make matters worse, many MSRs are not publicly accessible, and of the available MSRs, very few parameter sets (e.g. different corpora, different sensitivity parameters) are offered. For example, ICAN (Lemaire and Denhiére, 2004) is a well-founded MSR that one may implement, but no public ICAN service exists. PMI is a popular and easy-to- implement measure, but you would be hard-pressed to find a PMI service based on a news corpus, or an email corpus, etc. The MSR Web Server is an ongoing effort to gather various MSRs and corpora, to make these publicly available, and to give researchers easy standardized access to semantic relatedness scores from all MSR-corpus pairs.

General distance between to objects based on a list of attributes can be calculated based on following formula:

$$D(X,Y)=\sqrt{\sum_{\forall i, x_i\in X, y_i\in Y}\omega_i\cdot d(x_i,y_i)^2}, \tag{5.14}$$

where $d(x_i, y_i)$ is a distance by certain attribute and ω_i is weight for attributes. The requirement for the weights is:

$$\sum_{i=1}^{n}\omega_i=1, \tag{5.15}$$

where n is a number of attributes.

Now we have the same problem as in case of time interval distance measurement. We have distances between the resources based on certain attributes separately. The centers of masses those sets are not balanced and differently influence on result. Again, we have to modify the function with a correction part that balances centers of masses.

$$D(X,Y)=\sqrt{\sum_{\forall i, x_i\in X, y_i\in Y}\omega_i\cdot\left(0.5+\frac{0.5\cdot\left(d(x_i,y_i)-d_i'\right)}{\max\limits_j\left|d(x_i,y_i)_j-d_i'\right|}\right)^2}, \tag{5.16}$$

where d_i' is median of corresponding samples of distances by i-th attribute.

Then closeness of two resources (objects) equals:

$$closeness=1-D. \tag{5.17}$$

5.3 SmartInterface: MetaProvider architectures

GUI Shell allows user dynamic switching between MetaProviders for more suitable information representation depending on a context or resource nature. From other side, MetaProvider plays fore main roles:

- Context-aware resource visualization module that presents information regarding to specified context in more suitable and personalized for user form (see Figure 5.9);
- Interface for integrated information visualization with intelligent context-aware filtering mechanism to present only relevant information, avoiding a glut of unnecessary information;
- Visual Resource Platform that allows resource registration, search, access and modification of needed information/data in a space of registered resources;
- Mediator that facilitates resource to resource (R2R) communication.

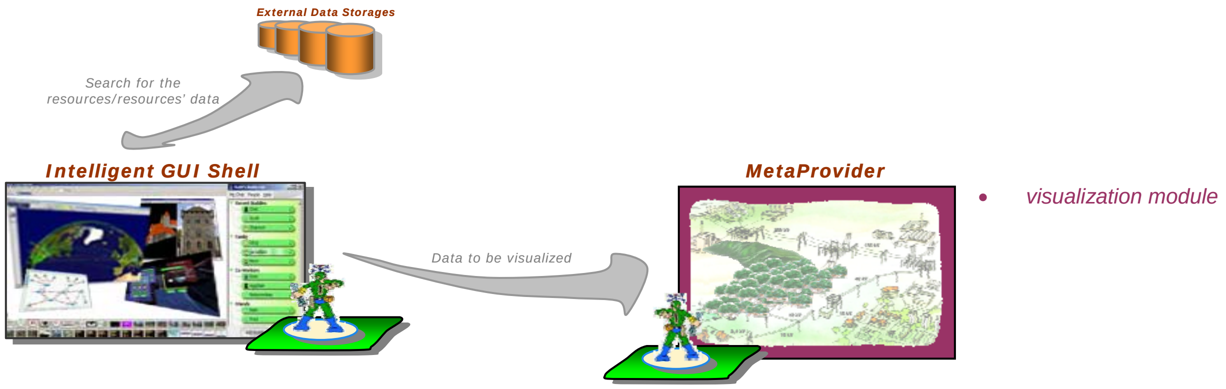


Figure 5.9 – MetaProvider - visualizator.

As a portal for registered resources, MetaProvider should support a simple resource registration process (depending on a specific of presentation view), store resource semantic profiles for further resource discovery and inter-resource communication. Such communication implies an observance of the same ontology by resources. Following the common ontology, resources have a possibility to create a correct request to each other, and get and present appropriate data. Integrated data representation means representation of the data from different resources in certain context related to a subject resource. It is a representation of filtered relevant to the certain case data. There is a sense to perform a filtering process on the user side (i.e. on the side of GUI-Shell), but an input data for the request creation and filtering process should come from the resources that are accessible via MetaProvider API. Depending on a context for required resource visualization, decision making module collects and send only relevant data to MetaProvider (see Figure 5.10).

In case, when selection of relevant resources and their properties depends on MetaProvider realization and specifics of visualization context, it is also reasonable to leave a decision making part on MetaProvider side (see Figure 5.11). Thus, MetaProvider collects necessary data from the registered resources, from the resources that are registered on other MetaProviders/portals, and from the external data storages with a help of GUI-Shell.

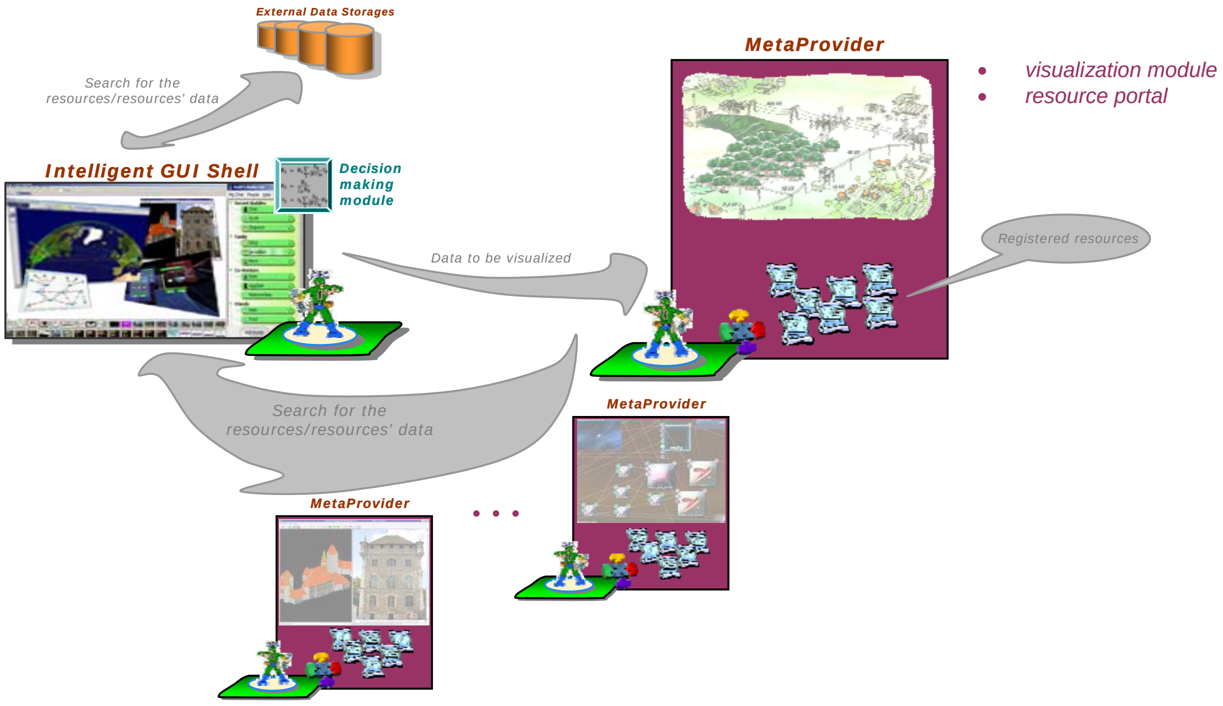


Figure 5.10 – Visualization and provisioning of data.

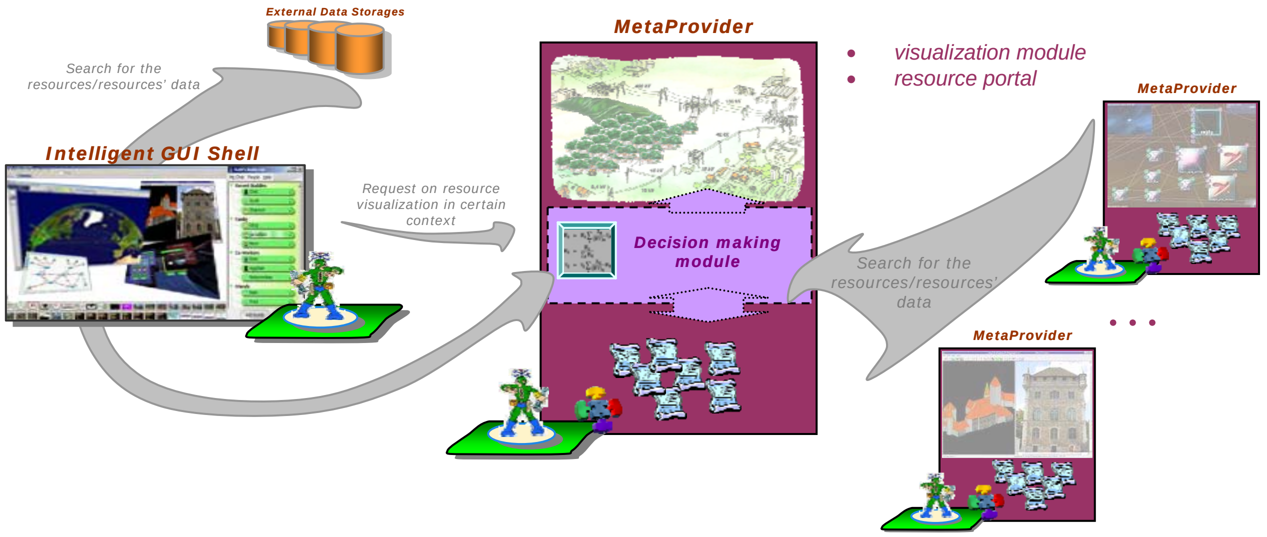


Figure 5.11 – Context relevant data collecting and visualization.

Regarding interoperability among the components of the environment, it should be provided via metadata and common ontology. Additionally, if we consider the open environment, members of which are agent-driven (i.e. proactive, autonomous, goal-driven, intelligent and etc.) to enable communications, coordination and negotiations between each other, then MetaProvider should support inter-agent communication via semantic API.

We consider MetaProvider as a universal visualizer that deliver the context information both to humans (in a visual form) and to agents for further processing (semantic data). MetaProvider should also provide context-based “visualization” for other resources (for agents that present them). Thus, relevant resources for subject resource visualization in certain context can be requested by some resource agent and decision making module of MetaProvider should “visualize” this information to the agent by sending required data to it (see Figure 5.12).

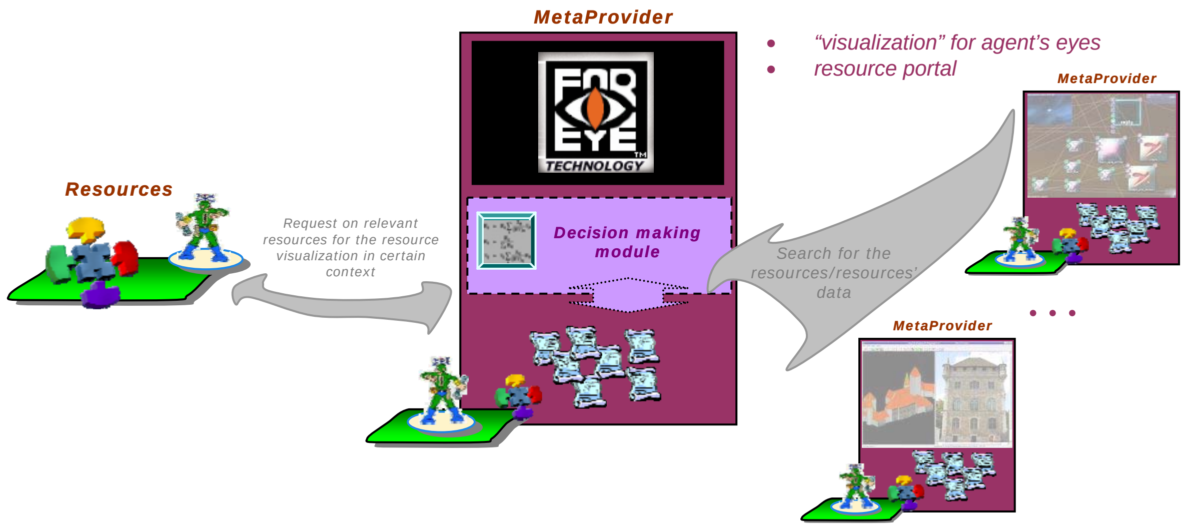

MetaProvider
“visualization” for agent's eyes
resource portal
MetaProvider
FOR EYE TECHNOLOGY
Resources
Request on relevant resources for the resource visualization in certain context
Decision making module
Search for the resources/resources' data
MetaProvider
. . .

Figure 5.12 – “Visualization” for agent’s eyes.

Different MetaProviders are specialized and present the resources in certain domain. Such domains can target restricted set of the resources as well as wide. For example, one of the restricted domains can be anatomy and processes between the parts of the human body. In this case, MetaProvider can present the information based on 3D human body visualization and should be able to visualize the properties from anatomy domain. This task becomes more challenged in case of wide target domain. For example, if MetaProvider represents information vie spatial resource location view and covers all of the resources, then it should somehow visualize all the properties. Thus, the main requirement in this part is to be able to visualize the properties of the resources that are belonging to the covered domain ontology.

UBIWARE Deliverable D2.1:
Workpackage WP6:
Task T2.1_w6:
Workpackage leader: *Michal Szydlowski*

6 Middleware for Peer-to-Peer Discovery

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.

No work was performed in this WP during the Year 1.

The WP tasks for the Year 2 (The *Discovery* phase) are the following:

Task T2.1_w6 (research): Design of a P2P resource discovery infrastructure for UBIWARE, including related inter-agent communication protocols.

Task T2.2_w6 (development): Incorporating the research findings to the UBIWARE prototype.

6.1 Background

With the growing popularity of Multi Agent Systems, the number of deployed multi-agent platforms will continue to increase. We see the necessity of existence of mechanisms which allow creating connections between these platforms. We believe that UBIWARE should be extended with functionality which would provide agents issuing search request not only with local results, but also with global results from the entire network of interconnected platforms. Our goal in workpackage 6 was to design a middleware that extends the UBIWARE functionality so that it could handle inter-platform communication and discovery in peer-to-peer manner. Such extension will allow us to create connection between separate UBIWARE systems where agents residing on different platforms could exchange information and cooperate.

6.2 Agent Platform architecture overview

This part describes the general architecture of an Agent Platform with its key components, which take part in agent communication and agent discovery.

AP provides the physical infrastructure in which agents are deployed and consist of the machines, operating systems as well as agent management components and agents themselves. The figure below (Figure 6.1) presents the general idea of the agent platform architecture.

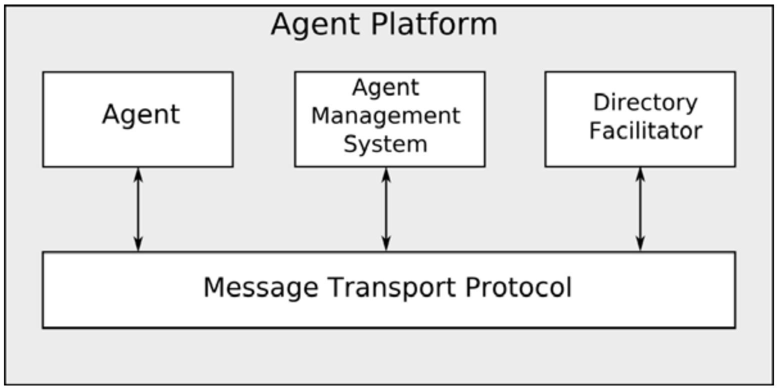


Figure 6.1 - Agent Platform architecture overview.

Following description of presented AP components (FIPA, 2004)(Bellifemine et al., 2007):

Agent – the computational process that is created in and inhabits an AP, typically offering services which after being published can be searched and used by other agents. An agent must have at least one owner (AP) and must be identifiable within a platform by Agent Identifier (AID) which allows an agent to be distinguished unambiguously trough a collection of parameter-value pairs and consist of

- **name –** unique identifier constructed of the actual nickname of the agent and its home agent platform address (HAP), separated by the @ character. For example John@platformhost:1099/JADE
- **addresses –** list of transport addresses where the message can be delivered. Since an agent may support many methods of communication, multiple values of physical

addresses can be contained within this parameter. An example of transport address could be *http://plaftormhost.UBIWARE.jyu.fi:7778/acc*

- **resolvers –** list of name resolution service addresses which can be utilized the name resolution service provided by AMS. This parameter is a sequence of AIDs describing AMS instances on different platforms.

Agent Management System (AMS) – a mandatory component, which is automatically created during the startup of AP and is responsible for its operation such as creation and deletion of agents. Each agent present on AP must register with AMS, which holds the directory of all agents residing on the platform, thus providing the white pages services allowing peer-to-peer discovery of agents via its *search* function. The AMS has always the same reserved name, which is *ams@plaftorm_name* (for example ams@platformhost:1099/JADE)

Directory Facilitator (DF) – the optional component of an AP. By maintaining the accurate and complete list of agents its task is to provide the yellow pages services to other agents. Directory facilitator must provide following functionality:

- register – an agent publishes its service description
- deregister – removing the description from DF
- modify – update of the service description
- search – query DF and obtain the list of agents playing certain role specified in the search criteria

The agents can register to the DF, publishing the description of the service it provides, thus allowing themselves to be discovered by other agents who are interested in such service. One AP may contain many DF allowing them to register with each other, thus creating federations.

Message Transport Service (MTS)[3] – manages FIPA compliant ACL messages exchange within one platform (intra-platform) and between platforms (inter-platform). Intra-platform communication in JADE takes place over Internal Message Transport Protocol (IMTP) within single container, and inter-container messaging uses Remote Method Invocation (RMI). The inter-platform communication is conducted trough Agent Communication Channel (ACC) which may employ different Message Transport Protocols (MTP) such as HTTP[4], WAP[5], IIOP[6], or JMS(Curry et al., 2003). The messaging architecture is illustrated on Figure 6.2.

During the platform initialization HTTP-based MTP is started by default. This creates a server socket on the host where main container resides which then listens to incoming connections at the URL *http://hostname.domain:port/acc* (example http://platfotmhost.UBIWARE .jyu.fi:7778/acc). JADE allows that any number of MTPs can be activated on platform; moreover a MTP can be added at run-time which enhances platform's communication capabilities (for example to locate and send message to an agent residing on mobile device).

[3] "FIPA Agent Message Transport Service Specification", 2002.
[4] "FIPA Agent Message Transport Protocol for HTTP Specification", 2002.
[5] "FIPA Agent Message Transport Protocol for WAP Specification", 2002.
[6] "FIPA Agent Message Transport Protocol for IIOP Specification" 2002.

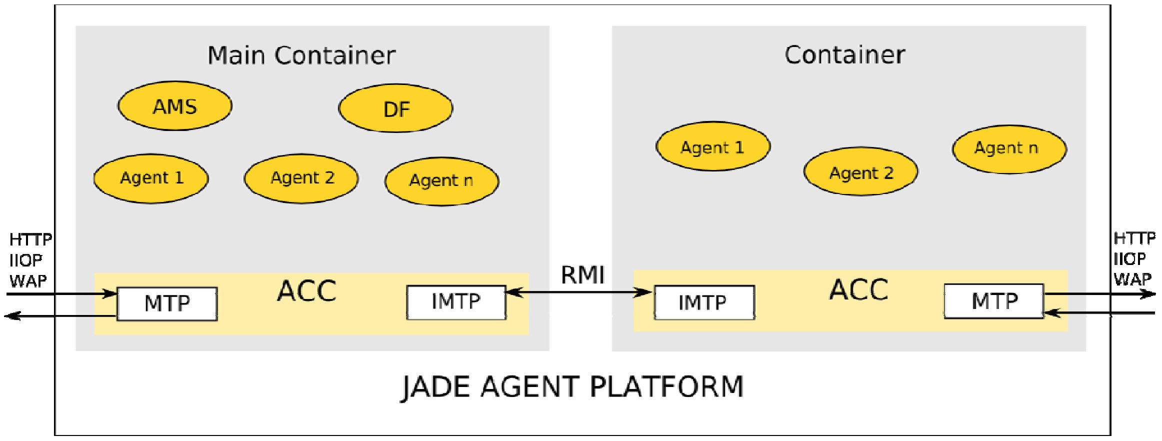


Figure 6.2 - JADE messaging architecture.

6.3 Inter-platform and agent discovery in multi-platform environment

Let us analyze the situation, where several independent UBIWARE-based agent platforms exist in the network, each having set of agents running and registered with DF.

Within each platform, inter-agent discovery and communication is not a problem, since the platform's components take care of these acts. The question arises, would be possible (and how) to allow the agents from different platforms to search remote DFs and interact with agents located on foreign platforms. Such solution deployed in UBIWARE platform would enhance its functionality – if agent cannot locate a service or obtain needed information within its home platform, it can query the DFs on known remote platforms in search of needed data. Agents would benefit from this possibility in two situations:

- The service or information required by an agent cannot be located within native platform,
- An agent possessing such service/information no longer exists within the platform or cannot be contacted.

We see the similarities between peer-to-peer file sharing networks and multi agent systems. In P2P networks the nodes exchange messages in order to locate their neighbors and resources they share. In multi agent systems the agents attempt to discover other existing agents and services they provide in order to utilize them, thus enhancing the overall functionality of the platform. In this section we analyze the existing approaches used to build file sharing P2P networks that could be utilized to address the problem – the centralized approach of general Directory Facilitator and solutions known from pure peer-to-peer file sharing networks with connection to JADE's mechanism of DF federations.

6.3.1 Central Directory Facilitator

This section introduces the centralized approach for inter-platform agent discovery. It borrows from the hybrid model known from file sharing peer-to peer networks implemented in Napster platform[7]. In hybrid model the central server stores a list of available resources each node is sharing. Moving this idea to multi-platform environment, one of the existing platform is designated to be a central server. Every agent wishing to publish its services contacts the central server and registers with the general Directory Facilitator. To discover an agent playing certain role, agents query the central server, which will return the list of addresses (AIDs) of agents providing required service. Figure 6.3 shows the idea of multi-platform environment with central Directory facilitator.

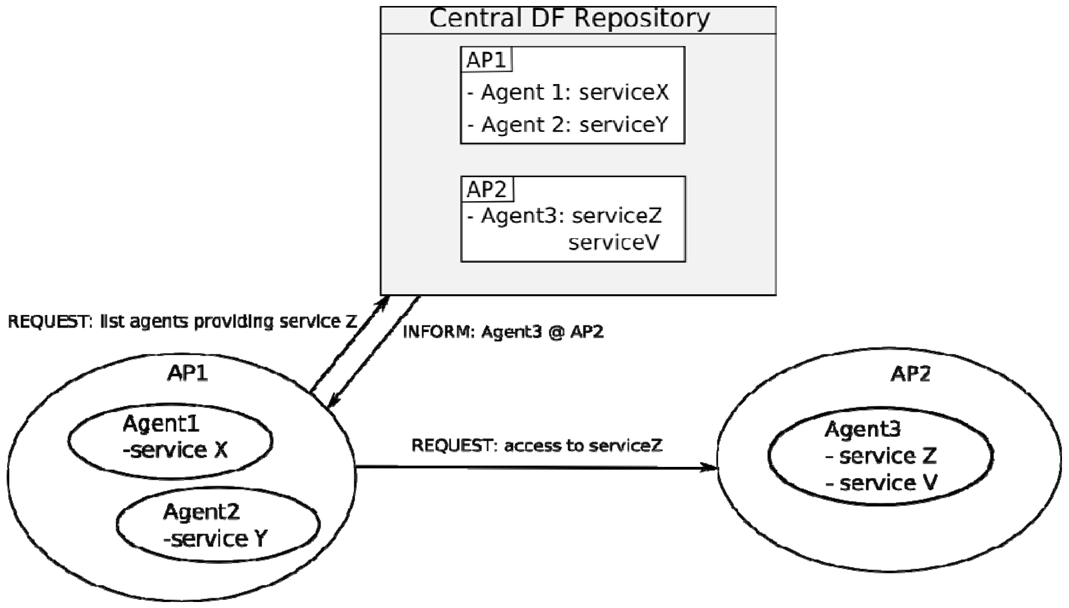


Figure 6.3 - Multi-platform environment with centralized DF.

However having the advantage of quick response time to issued query, the centralized approach suffers from two drawbacks. In the dynamic environment the central Directory Facilitator will have to rely on agents updating their information to keep the repository up to date. The server has to process these updates in addition to handle incoming service discovery queries, which increases load on the server and requires high maintenance to host it. Second drawback relates to reliability of centralized approach. If the central Directory Facilitator becomes for some reason unavailable, the agents loose ability to publish and discover services among the network. To improve the survivability of multi-platform environment, we must provide the eventuality of platforms communicating with each other without relying on central server. In the next section we examine such mechanism of creating persistent connections between Directory Facilitators.

6.3.2 Federated Directory Facilitators

This section introduces a notion of Directory Facilitators federation. Following FIPA specifications, JADE enables Directory Facilitators to register with each other, creating federation within a single (also distributed) platform. When DF agent registers with another one, it becomes its child, and will from this point receive all search queries from its parent DF, also forwarding it to each of its child. The propagation of the query within federation is

[7] Napster platform - http://opennap.sourceforge.net/napster.txt

controlled by a parameter *MaxDepth* in search constraints, which specifies how many times the query should be forwarded. The parameter *MaxResults* limits the number of agents' descriptions that will be returned to agent issuing a query. Figure 6.4 shows an example of Agent Platform with multiple federated DFs each having set of agents registered with it.

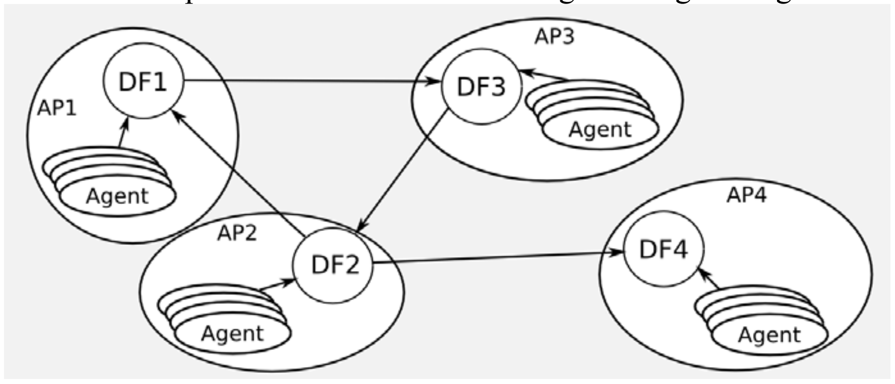


Figure 6.4 - Federation of Directory Facilitators.

The direction of an arrow between DF indicates the parent-child relation in this case: DF1 is parent of DF2, and is also a child of DF3 and so on. The links between the nodes in federation are unidirectional – the parent propagates the query only to its children, never to its parent(s). Let's examine a situation, where Agent1 subscribed to DF1 is searching for agents who can provide him with *serviceX*. We assume that *serviceX* has been published by Agent3 registered with DF3. The query is submitted to DF1, which propagates it to its child – DF2. DF2 then propagates it further to DF3, and the results are sent back since there is no other node to propagate the query to (same when the value of parameter *MaxDepth* has been matched). Agent1 issuing the query is given the results by means of AID of the Agent3, and can now contact it directly and request to use its service.

The federations can be useful feature provided by JADE, to create a network of connected agent platforms. However this mechanism has also some drawbacks, which have to be mentioned. First, due to unidirectional link created between parent and a child, the query is propagated always in one direction only. In example presented above if the agent publishing *serviceX* was registered with DF4 instead of DF3, it would not be discovered by Agent1@DF1. To address this problem, the possibility of bidirectional link between Directory Facilitators must exist. This could be achieved by forcing each DF that is receiving the request for registration to automatically register with requestor.

Another assumption made in the example was that the connections between the nodes already exist. This requires the topology to be built based on the nodes' addresses provided in platform's code or configuration files. Such solution is problematic, as it requires high-maintenance when the topology is modified and the nodes require to be started in certain order. Also if any of the connected nodes loses connection to the network, the topology has no ability to automatically recover from this failure. An agent platform that uses discovery is not restrained by these problems. In the next section we examine the Gnutella[8] protocol which represents the mechanism of creating pure peer-to-peer networks.

[8] Gnutella - http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

6.3.3 Creating dynamic peer-to-peer topology

Gnutella is a decentralized peer-to-peer system, which means it operates without presence of centralized server. Each participant acts as both server and a client; therefore the term "servent" was introduced. All servents share resources allowing others to locate and download them. By using messages passing system all servents execute two types of operation. First, by exchanging messages servents increase or maintain their degree of connectivity. Secondly, the search queries are exchanged in order to locate required resources that may be shared by some servents connected to the network. The protocol requires that within the network exists at least one always-on node, which provides a new participant with addresses of the servents already operating. Each servent upon startup obtains a pool of addresses and connects to them. In order to discover other participants it starts the *PING* / *PONG* process, presented in Figure 6.5.

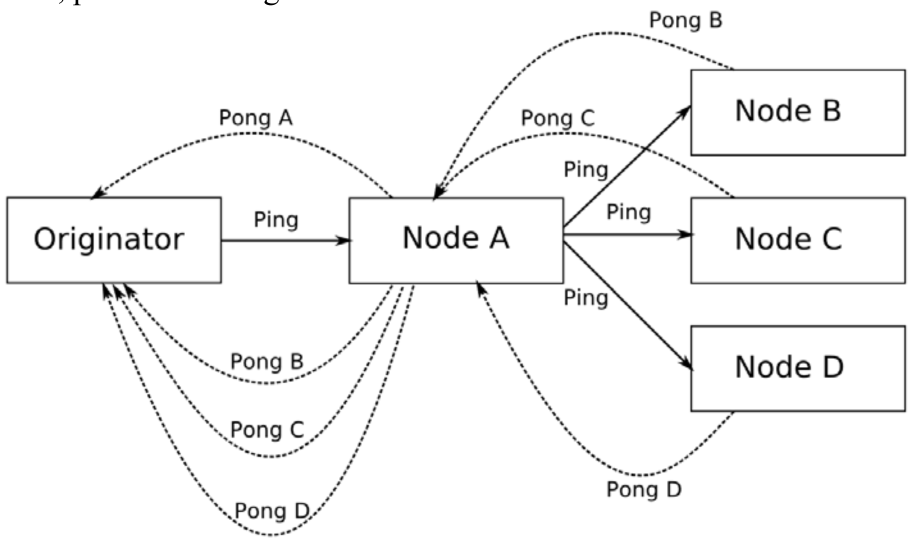


Figure 6.5 - Propagation of *PING* and *PONG* messages in Gnutella discovery process.

The originator sends *PING* message to every node it is connected with. The *PING* message contains TTL (Time To Live) parameter which specifies the number of hops the message should be forwarded, until it stops to propagate. The originator awaits then incoming *PONG* messages that will contain address of node and overall information on resources shared. Upon receiving the *PING* message, a servent performs two actions: first it sends back the *PONG* message along the path the message came to ensure the respond reaches the originator. Secondly it forwards the *PING* to known servents with reduced TTL. The *PING* message stops to propagate as soon as TTL reduces to 0. Due to high connectivity between nodes on Gnutella network, the *PING* message can reach an exponential quantity of servents. If we consider an example, where each servent maintains 4 active connections, and the issued *PING* message had TTL of 5, the message could reach as many as 1405 servents on the network. During its lifetime, a servent will also receive *PING* messages from new participants entering the network that were not online when it first performed its discovery process. The new addresses obtained from incoming *PING* messages can be then stored and used to create new connections in the future, for example when some of known servents are not available anymore. The process of harvesting new nodes' addresses brings also significant benefit to the network – the routes between servents will have more of resemblance to graphs than to trees, therefore even if some servents do not route the messages, it is likely that the message will be transmitted over an alternate route. Gnutella's search mechanism uses the same process of message passing that is used to discover other servents. The only difference is that the servent routes the search query first, before performing more time-consuming local search.

6.3.4 Building peer-to-peer network of agent platforms

The operating process of Gnutella's distributed network of servents is very similar to the mechanism of federated directory facilitators described in section 6.3.2. The federated DFs forward the query by one hop, as servents do. The query propagates within the federation for certain amount of hops, specified by the *MaxDepth* parameter, while Gnutella uses TTL to control the propagation of *PING* and Query messages. If we image the agent platforms being the nodes in the network and the directory facilitators to be the servents, it is then safe to assume that we can employ the ideas behind pure peer-to-peer approach to create middleware for inter-platform discovery. To achieve this goal, we propose several modifications to the existing DF structure to form a distributed network with other DFs in other agent platforms. From now we will refer to a single agent platform with directory facilitator and agents as node. We propose following modifications:

- Bidirectional links between the platforms while federating directory facilitators. This will ensure building graph resembling topology in the network.
- Before routing the message to the neighbors, the node checks the originator of the message and block forwarding it back. This enhances the performance by eliminating the duplicate queries and limits the query flooding the network, lowering the unnecessary traffic.
- Each node has set of *n* active bidirectional connections at a time. Additionally, a node can maintain a set of *m* children connected trough unidirectional links.
- The node is capable of storing locally new addresses of other nodes. The addresses are harvested from incoming search messages. New addresses can be used to create future connections (federations) in situations where some of existing links became broken. This enhances the connectivity within the network and improves its survivability.
- Within the environment there exists at least one server node that is capable of storing addresses of the nodes currently available on the network. The server accepts registration requests from new participants and provides them with access to pools of addresses of active nodes.

Based on these modifications, we can now sketch the scenario for inter-platform peer-to-peer discovery in multi platform environment.

6.3.4.1 Startup

Upon startup, the node contacts one of the always-on servers and register with it. The registration information consists of node's address and value of *validityTime* parameter specifying the amount of time after which the node has to re-register. This is done to ensure that the server has always the most recent data of existing nodes within its repository. The server stores the node’s address received during the registration process to provide it to other new nodes.

6.3.4.2 Discovery

Next the new participant requests a pool of i (where $i \leq n$) addresses from the server. The server randomly selects i addresses from its register and sends it to requestor. Having received the set of addresses, the node now sends registration request message to obtained addresses. The recipient of the message can accept the request if the amount of its connections is less than n. Otherwise if the recipient cannot create new connection, the request is denied, however the request sender can be added to a set of recipient’s children. Creating such “weak” (unidirectional) link will allow the node to receive query messages and helps to avoid the situation, where the new node entering the network stays isolated being unable to connect with neighboring nodes.
Upon successful registration between the requestor and the recipient, a bidirectional link between the two nodes is created. After this process, if the node has still empty slots for creating new connections, it can request a new set of addresses from the server. This can be repeated until the node creates desired number of connections. Apart from providing the requesting node with a set of addresses, the server does not play any role in service discovery.

6.3.4.3 Searching for resources

The process of registration and resources lookup between nodes is handled by exchanging ACL messages. An agent who wishes to discover certain service issues a query to his own directory facilitator. The query contains of a description of the service the agent is interested to locate along with parameters *MaxResults* and *MaxDepth*. The *MaxResults* parameter informs how many discovered agents’ information the query sender wishes to get in reply. *MaxDepth* parameter specifies the number of hops the query should be propagated. Therefore by adjusting this parameter the agent can decide the range in the network its query will reach. The local search query has the *MaxDepth* parameter set to 1, and allows agent to quickly discover agents from his home platform. For the systemic queries, the process proceeds similarly to searching in Gnutella network; each node upon receiving the query decrements the value of *MaxDepth* parameter by 1, forwards the query to neighboring node and then performs the local search. The query is forwarded until the value of *MaxDepth* reaches value of 0. Then results of each local search are combined as the respond travels back the path it has been routed until they reach the originator.

6.3.4.4 Recovering from lost connections

In a dynamic environment of multiple agent platforms the nodes must be able to react to changes that may occur in the topology of the network. Each node should have the mechanism of active maintenance of own existing connections along with discovering new platforms to connect to. As the current participants may become unavailable, the node may find itself being isolated. One of the instruments to tackle this problem is repeating the discovery process described above, where node requests a new set of addresses from the server. In addition to that we may utilize the process of locating services on remote platforms to obtain new addresses to connect to. During the search process, the search response travels back to the originator. Upon successful search, the platform will receive the AID of a remote agent providing the desired service. The AID contains of the agent's name and the address of the remote platforms. This address can be then used to create new connection extending the network graph. Figure 6.6 illustrates an example of this process.

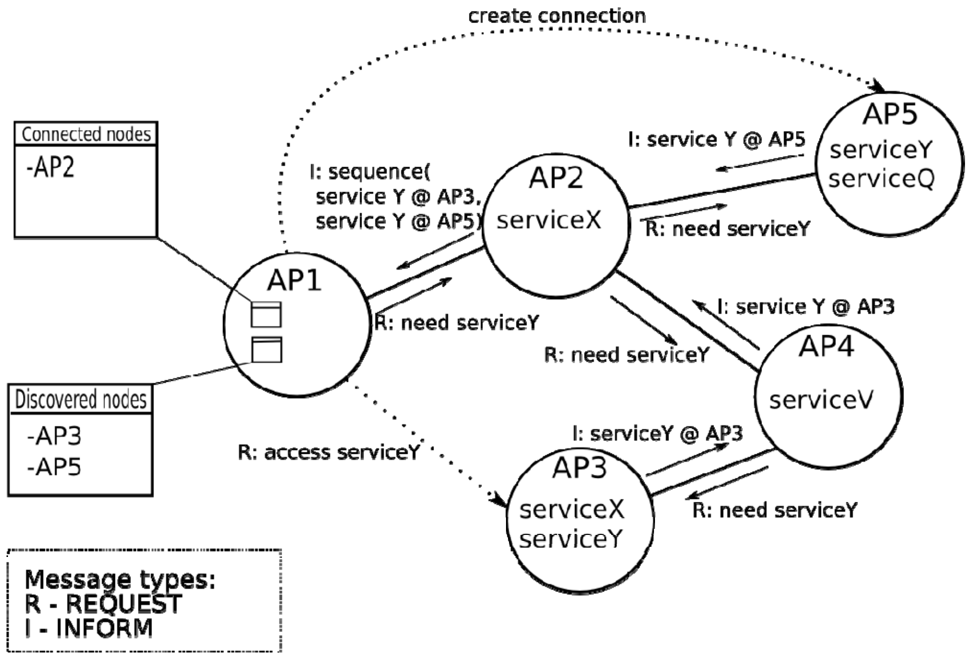


Figure 6.6 - Discovering new nodes after successful service search.

In this example we see a set of remote platforms, already having connections between each other. A request to locate service Y is issued by an agent residing on AP1, which ha only one neighbor. Following the search process, the query floods the network and the results are routed back along the path the query travelled. The query originator receives AIDs of two agents, located on platforms AP3 and AP5, so he can request an access to the service directly from the remote agent. As the query results are processed by agent's home directory facilitator, it can process them to extract the addresses of new platforms. These addresses are stored in the *Discovered Nodes* repository and used to create new links.

One may notice in the example above, that query flooding the network to reach the distant nodes and being routed back produces a significant amount of traffic. We can tackle this problem by improving the querying and not have the query travelling back the same path it has been routed. Instead, only the nodes that contain searched service are replying directly to the query originator. This speeds up the search process and conserves the network's resources.

We have shown that based on federation mechanisms available in JADE platform and employing methods of building peer-to-peer networks between directory facilitators, we can create a topology where agents are able to discover services published on remote platforms. Such solution will lead to building a network of interconnected platforms where the search query will reach distant nodes not connected directly to node issuing the search. However, due to randomness of links created between platforms, such network is unstructured. A platform can be connected to its neighbors who don't publish required services, thus their only function is to route the search query. We see a field for further improvements in the process of building peer-to-peer topology. Allowing a platform to decide which links are useful and dropping the connections that do not provide satisfactory results we add the mechanism of creating clusters within the network where nodes are connected with the neighbors that are more likely to provide the successful search results. We describe the algorithm of creating these clusters along with further modifications in the next section.

6.4 Actively Adapting Topologies

In this section we present further assumptions and modifications to the Directory Facilitator implementation that lead to creating sets of connected nodes grouped by similarities between them.

Platform Profiling

In addition to assumptions presented in section 6.3.3 we introduce further modifications to presented approach. Each platform is described by its profile, which is set of information such as geographic location, company that runs the platform and general functionality. Upon startup, during the process of registration with central server, the platform provides also its profile. The server stores platform's address along with profile and instead of randomly choosing the pool of addresses sent back, it tries to find within its repository nodes with similar or required profiles. This modification may lead to creation clusters of connected platforms, where nodes are more likely to find desired services and is slightly similar to pure cluster approach presented by (Srinivasan and Finin, 2002).

Self-improving topology

Recent research in a field of file sharing peer-to-peer networks focused on creating mechanisms that would allow nodes to actively participate in improving the network's structure. It resulted in creating several algorithms for topology management such as APT algorithm presented by (Condie et al., 2004), Node Selection Algorithm, Node Removal Algorithm or Overtaking Algorithm presented by (Kamvar et al., 2003). We consider the APT algorithm to be the most suitable topology management algorithm for our case. We take a closer look at the APT algorithm to show how the general idea behind it can be applied in multi-platform environment.

In their work Condie *et al* base Adaptive P2P Topologies (APT) on two notions:

1. A peer should directly connect to those peers from which it is likely to obtain resources in the future.
2. A peer may use its past history to estimate the likelihood of a future successful download.

In the APT protocol, a peer encodes its past history with a *set of local trust values* (Kamvar et al., 2003). Peer *i* stores a local trust value for each peer it has interacted with. If *sat(i, j)* is the number of satisfactory transactions peer i has had with peer *j*, and *unsat(i, j)* is the number of unsatisfactory transactions peer *i* has had with peer *j*, then we define the local trust value as

$$s_{ij} = sat(i,j) - unsat(i,j), \qquad (6.1)$$

Peer i may deem a transaction unsatisfactory if, for example, the communication with peer j cannot be established. The local trust vector s_i associated with peer i contains all s_{ij} values where j varies over all peers in the network. In their implementation, each peer maintains a hash table containing the local trust values of all its acquaintances. An acquaintance is then defined as an entry in the hash table. If peer i has never interacted with peer j then there will be no entry in peer i's table for peer j, and thus $s_{ij} = 0$.

The APT protocol is a peer-level greedy algorithm and it proceeds as follows. Peer i joins the network by attempting γ random connections. The join process is repeated until at least 1 connection is made. After successful transaction with peer $j \in N(i)$ where $N(i)$ is a set of peer's active connections, peer i with $n_i < \tau$ (where τ is maximum amount of connections for peer i) connections sends the connection request $R(i, j)$ to peer j. If $n_i = \tau$ then peer i sends a connection request if one of the following holds:

1. Peer $j \notin N(i)$ achieves a higher local trust value than one of peer i's neighbors (message $R(i, j)$ is sent).
2. Peer $j \in N(i)$ is assigned a lower local trust value than some acquaintance $k \notin N(i)$ of peer i (message $R(i, k)$ is sent).

The first scenario describes an successful transaction with peer j, while the second occurs after an unsuccessful transaction with peer j. In both cases, if peer i's connection request is granted then it will disconnect from its neighbor with the lowest local trust value. A special case occurs if a neighbor of peer i is assigned a negative local trust score and peer i is not able to connect to an acquaintance. In this situation peer i immediately disconnects from that neighbor and attempts a connection to a random peer.

Peer j will only accept peer i's connection request if peer j's local trust value s_{ji} of peer i is non-negative and one of the following conditions is true:

1. Peer j has fewer than τ connections.
2. Peer j's local trust value s_{ji} of peer i is greater than the local trust value of at least one neighbor.

In the second case peer j will replace its lowest trust valued neighbor with peer i.

We have shown the similarities between file sharing P2P network and environment of multiple multi agent platforms in section 6.2. These similarities allowed us to adapt the mechanism of building peer-to-peer networks of interconnected nodes to create mechanisms for building infrastructure of interconnected multi agent platforms. We believe that this applies also to algorithms designed to dynamically improve the topology of P2P networks, thus we can use the APT algorithm. We then propose further modification in addition to these presented in section 6.3.3. Except keeping the repositories of connected and discovered platforms, the directory facilitator should also map the results of locating agents and services on remote platforms to their addresses. Each time the directory facilitator receives the successful (not empty) query result it increments the score of the corresponding connected or discovered platform (value of s_{ij}) in adequate register. Then the APT algorithm is applied as described above to determine whether the directory facilitator should modify its existing connections.

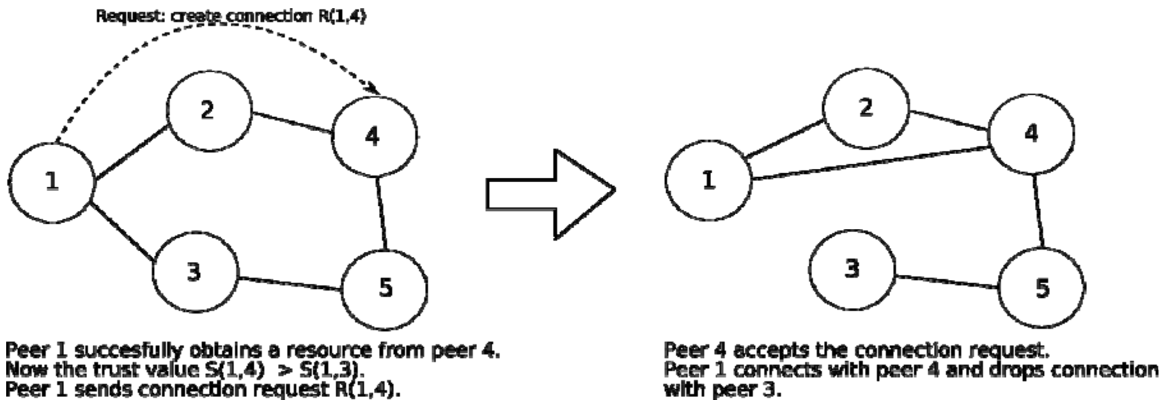

Request: create connection R(1,4)
1
2
3
4
5
Peer 1 succesfully obtains a resource from peer 4.
Now the trust value S(1,4) > S(1,3).
Peer 1 sends connection request R(1,4).
Peer 4 accepts the connection request.
Peer 1 connects with peer 4 and drops connection with peer 3.

Figure 6.7 - Example of utilizing the APT algorithm to adjust the topology.

In previous sections we were approaching the problem of peer-to-peer resource discovery from rather technical point of view. We observed the mechanisms of creating peer-to-peer topologies in file sharing networks and based our proposals on adaptation our observation to multi-agent environment. In next section however, we look at the multi-agent system as at social network, and present our findings on resource discovery in social peer-to-peer networks.

6.5 Social networks and resource discovery

Ramakrishnan and Tomkins estimate that publicly visible text creation of mankind is around 10 Gbytes per day, while private text creation is about two orders of magnitude higher at 3 Tbytes per day. Their upper bound on text creation is roughly another two orders of magnitude higher at 700 Tbytes per day. By 2010, buying storage to hold all global textual output would be financially equivalent to maintaining 10 people on payroll. (Ramakrishnan and Tomkins, 2007)

Information discovery through search is and will remain the driving force behind rich patterns of access to Web content. However, Ramakrishnan and Tomkins envision that search will change significantly. Much content of interest to users, both at the level of serendipitous consumption through a network or recommendation, and at the level of targeted information discovery via a search engine, will be created by other users rather than professionally produced. The objects to be retrieved will have a structure, an ownership, nontrivial access-control restrictions, and a broad range of heterogeneous metadata gathered from many sources. (Ramakrishnan and Tomkins, 2007)

Although global objects and portable contexts can realistically be hosted in a distributed manner, even in today's relatively simple search ecosystem there is no credible distributed search platform at the scale supported by the major search engines. Further, the Web almost completely fails to address the critical problem of access control. Major search engines index the public web and leave the rest untouched, allowing individual sites to provide search over private content as they see fit. Information will probably be meaningfully restricted to groups that range in size from two to tens of thousands or even larger. It is not feasible to first load all content matching a query and then restrict access, as today large portion of online content is private. Search over all relevant private repositories, which could easily number in the tens of thousands, is likewise impossible. (Ramakrishnan and Tomkins, 2007)

6.5.1 Social Networks

Wellman (Wellman, 1997) defines *social networks* as groups of people in a social setting or an organization connected by relationships. Mitré and Navarro-Moldes (Mitré and Navarro-Moldes, 2004) define a social network as a group of people, known as the actors, linked together by some pattern of interaction. These networks are normally represented by graphs called sociograms. A sociogram is a set of points (or vertices) denoting people, joined by lines (or edges) denoting links.

The "Six Degrees of Separation" phenomenon, i.e. that everyone is an average of six social links away from each person on Earth, was first investigated by Stanley Milgram in 1960 when he addressed letters to a particular stockbroker in New York and gave them to people randomly picked at locations in the United States far away from that of the final receiver. The condition for passing the letter, so that it reaches the final receiver, was that one could post it only to people they knew personally by first name. Eventually most of the letters reached the destination and the average number of hops was six. Since then there have been various studies demonstrating how this effect may help people conduct their everyday lives. (Anwar et al., 2004)

Power-law networks are networks where the probability that a node has k neighbors (called degree) is proportional to k^{-y}, for large k and $y > 1$. The parameter y is called the power-law coefficient. (Mislove, 2007)

Scale-free networks are a class of power-law networks where the high-degree nodes tend to be connected to other high-degree nodes. (Mislove, 2007)

There have been a series of studies of the structure and topology of Internet-based networks best summarized in (Pastor-Satorras and Vespignani, 2004) including the WWW, Email, instant messaging, virus/worm propagation, and P2P networks. Before this work identifying and quantifying the scale-free nature of the Internet, every new algorithm proposed by researchers for improving network performance was typically tested on random networks generated by consensus tools (such as the Waxman Network Topology Generator) which in retrospect resulted in incorrect solutions which should be reexamined. (Anwar et al., 2004)

The average path length among Microsoft Messenger instant messaging users is 6.6 among 240 million people. The network degree distribution is heavy tailed but does not follow a power-law distribution. Using maximum likelihood estimation, the distribution fits a power-law with exponential cutoff $p(k) \sim k^{-a}e^{-bk}$ with fitted parameter values $a = 0.8$ and $b = 0.03$. A strong cutoff parameter and low power-law exponent was found suggesting a distribution with high variance. (Leskovec and Horvitz, 2008)

Homepage networks arise because it is popular for students to mention their friends on their homepages, and link to those friends' homepages if they exist. Adamic and Adar studied at all users having a homepage under the domains www.stanford.edu and {web,www}.mit.edu. These sites contain the homepages of students, faculty, and staff. (Adamic and Adar, 2003)

One may expect that friends should have the most in common, while friends of friends should have less in common (and so on). We see that this is indeed the case in the homepage network as shown in Figure 6.8. In this figure Adamic and Adar plot the average combined likeness score versus distance, taking into account text, links, and mailing lists. In line with their hypothesis, the result appears as a rapidly decaying function in which the likeness score quickly falls off as the distance increases. (Adamic and Adar, 2003)

Since individuals tend to organize both formally and informally into groups based on their common activities and interests, the way information spreads is affected by the topology of the interaction network. (Adamic and Adar, 2004)

The information any individual is interested in depends strongly on his characteristics. Furthermore, individuals with similar characteristics tend to associate with one another, a phenomenon known as *homophily*. Conversely, individuals many steps away in a social network on average tend not to have as much in common, as shown in Adamic and Adar study of a network of Stanford student home-pages. (Adamic and Adar, 2004)

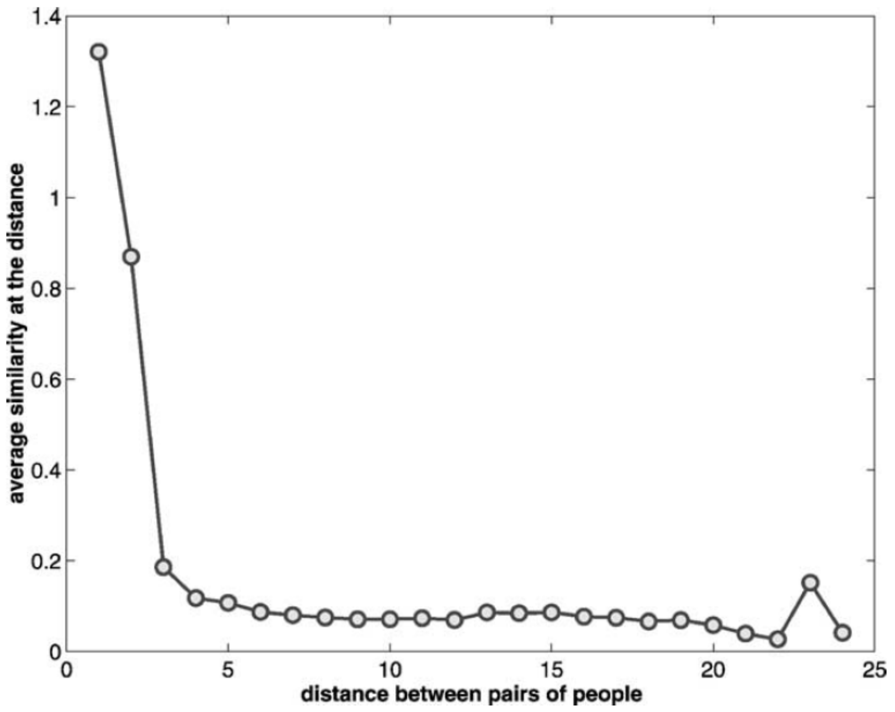


Figure 6.8 - The average likeness and the distance between individuals.

6.5.2. Social Networking Services

Social networking services (SNSs) provide an online private space for individuals and tools for interacting with other people in the Internet. SNSs help people find others of a common interest, establish a forum for discussion, exchange photos and personal news, and many more. (Ahn, 2007)

Several social networking services (SNSs) have been launched on the web. Participants in SNS can invite their acquaintances as new users to foster the social network. They can also register their personal information such as real names, living places, and hobbies to show to others. Besides, a friends list and communities are essential features of SNS. The list of one’s acquaintances is managed and published by the friends list. A community is a group of participants with common interest. (Okada et al., 2005)

Users publish content by posting it on a social networking site. Content is associated with the user who introduced it, and with users who explicitly recommend the content. Explicit links do not generally exist between content instances, and the content can be of any type. Often, the content is temporal in nature (e.g. blog postings), non-textual (e.g. photos and video clips), and may be of interest only to a small audience. (Mislove et al., 2006)

Independently from the content, users maintain links to other users. The links indicate trust or shared interest. Links can be directed (indicating that the source trusts or is interested in the content of the target) or undirected (indicating mutual trust or interest in each other's content). Some systems maintain groups of users associated with different topics or interests; users can then join groups rather than specifying links to individual users. In some systems, the full social network graph is public; in other, only immediate neighbors of a node can view that node's other neighbors. (Mislove et al., 2006)

The impact of SNS on Internet traffic is significant. HitWise (www.hitwise.com) announced in 2006 that MySpace SNS is the number one website in the United States. MySpace had been observed to receive 4.46% of all hits in U.S., more page hits than Google with 3.89%.

6.5.3 Social Peer-to-Peer Networks

Social P2P networks differ from file-sharing P2P networks, because the network topology is mainly formed from connections in a living social network of users rather than freely selected by topology management algorithms of a P2P network. These kinds of networks can be found for example in address books of mobile devices or centralized social networking sites. Research on resource discovery in social P2P networks is still in its infancy – simulation results from a resource discovery algorithms applied in social networks simply do not exist.

Resource discovery problem in P2P networks assumes an underlying topology between peer nodes and resources. Then the task of a resource discovery algorithm is to locate a predetermined number of resource instances from the topology using local information stored in a peer, which is currently processing the resource discovery query.

In 1969, Thomas Shelling, an economist, proposed a model to explain the existence of segregated neighborhoods in America. He observed that the appearance of such segregated neighborhoods is caused neither by a central authority, nor by the desire of people to stay away from dissimilar people; instead, it is the cumulative effect of simple actions (moves) by individuals who want at least a certain proportion of their neighbors to be similar to themselves. (Singh and Haahr, 2007) Based on Shelling's findings it is also expected that social networks are searchable, because users with similar interests tend to connect to each other in social networks. Thus social networks naturally organize the data into different topics.

Usually, in social P2P networks, the starting point of a query is critical, because the information found from nearby peers is more relevant than from the peers further in the social network. Consider for example the problem of finding information about a certain company. Normally people use centralized web search engines like Google to find out information about the company. This approach returns information which is popular or useful for many people or otherwise it would not appear in the result list of a search engine. Now, consider we have a social P2P network between mobile devices and possibility to execute queries there. By executing a query starting from the mobile device of a person who searches for the information, the searcher can locate for example information who is the closest person in my social network who works in that company. Then by searching the contents of that peer it is possible to get information, which is not available from centralized web search engine.

Most of the architectural features common in file-sharing P2P networks are present in social P2P networks also. These features include low cost, fault-tolerance, scalability, load balance and real-time operation. Architecture has low cost, because investment for the servers is not needed. Fault-tolerance stems from the distributed nature of the network and the absence of a single point of failure. Scalability and load balance can be achieved with distributed algorithms avoiding processing cost bottlenecks found from server-based architectures. Real-time operation is achieved with local indexes in peer nodes and thus no delayed index updates or site crawling are needed.

The main benefits of social P2P networks' architecture compared to centralized search engines, e.g. Google, are in addition to low costs, real-time searching capabilities and a possibility to locate rare, but socially relevant data. An example of such data is information on who of my acquaintances have watched a certain movie or who of my colleagues have written about a certain topic. In addition to social relevance, other dimensions like contextual relevance can be used as a measure. An example of this is data located in nearby mobile devices.

The social networking services like Facebook (www.facebook.com), MySpace (www.myspace.com) and LinkedIn (www.linkedin.com) have become popular means to share personal information. People can for example live-broadcast themselves via justin.tv (www.justin.tv) or share photos through Flickr (www.flickr.com). Architecture of these services stores the data on centralized servers consuming network bandwidth and storage space. For example YouTube (www.youtube.com) stores lots of videos on the centralized servers but this architecture comes with high costs. According to Forbes (Forbes, 2006), in 2006 YouTube was estimated to transfer 200 terabytes of data each day with costs of one million dollars per month. Some P2P technology developers estimate that savings of 95% in hosting costs can be achieved with P2P architecture (Forbes, 2006).

6.5.4 Mobile Social Peer-to-Peer Networks

Another viewpoint is the on-going transition in the development of mobile devices. For a long time mobile devices have been isolated from the Internet. Calendar data, notes, photos etc. have been kept private in the mobile device and whenever needed transferred using short-range connections like Bluetooth to desktop computers for storage or sent via short messaging system (SMS) to other mobile users or centralized servers. However, during 2007 NOKIA has ported Personal Apache MySQL PHP/Python (PAMP) stack (opensource.nokia.com/projects/PAMP) to Symbian phones thus opening up the mobile device for external access from the Internet. Now it is possible to develop web services on a mobile device or provide access to different kind of content using the de facto tools of traditional web programming. This creates a huge and a rapid wave of mobile web application development, because programmers can now use their existing development tools.

It is expected that the current social network systems will evolve towards distributing the content of the centralized servers to the mobile devices. In a distributed architecture, the users of the mobile devices publish their content (files, folders, calendar etc.) locally in the devices via mobile web server and define the access rules for the content using the motto "I only display what I want to who I want". This can be realized for example using an indexing architecture similar to Google or Apache Lucene tools (lucene.apache.org), but now locally in a mobile device with an access-controlled search engine.

The social P2P networks' architecture allows for example developing a system similar to Facebook on a mobile device. This kind of social network applications have shown to attain a huge user base. Other uses for search functionality on a mobile device are applications for content sharing to small groups, friends or people interested in same hobbies. In a best case, the system can be used as a complement to Google search interface if content is new enough to not have been indexed by centralized search engines.

In general, the Mobile Social P2P networks architecture shortens the information flow chain significantly and complements centralized search engines. Now it takes a long time when captured information (e.g., photo) can be found from centralized search engines. With Mobile Social P2P networks' architecture the users who search data reveal which content becomes popular and this content can then be added to centralized search engines making it better searchable.

6.5.5 Resource Discovery Problem

There are various names for peer-to-peer resource discovery problem in literature. Menascé et al. (2002) use term *deterministic location problem* for the problem "given a resource name, find the node or nodes that manage the resource" and *probabilistic location approach* to address the problem of "given a resource name, find with a given probability the node or nodes that manage the resource". Kalogeraki et al. (Kalogeraki et al., 2002) define the *information retrieval problem in the P2P networks* as the following: Assume that each peer has a database (or collection) of documents, that is also made available to all peers connected in the network; this represents the knowledge of the peer. A node searches for information by sending query messages to its peers. We assume that the queries are collections of keywords. The querying peer is interested to find all the documents that contain all the keywords. A peer receiving a query message evaluates the constraint locally against its collections of documents. If the evaluation is successful, the peer generates a reply message to the querying peer which includes all the documents that correspond to the constraint. Once it receives responses from all the peers, the querying peer can afterwards decide which documents to download.

6.5.5.1 Resource Discovery in Peer-to-Peer Social Networks

NeuroSearch is a neural network-based resource discovery algorithm for P2P networks. The aim of NeuroSearch is to combine a set of resource discovery information to decide which of the neighboring peers are worth querying. The distinct feature of the NeuroSearch framework is that it can combine many types of resource discovery information into one query forwarding decision and thus may potentially encompass the good sides of different resource discovery algorithms. Currently NeuroSearch is the only P2P resource discovery algorithm combining multiple different local information types together. The drawbacks of the NeuroSearch framework are the computationally costs involved in the training of the neural network and possible inaccuracies of training data with the real-world P2P network environment.

NeuroSearch could be adapted to a social P2P network also. In a social P2P network node degrees can easily lead to thousand of neighbors and therefore e.g. BFS sending all first level neighbors at once cannot be directly used, the granularity of the search is too coarse. Random walk is also problematic because it is expected that in a social network nearby peers are likely to provide information about similar topics and therefore algorithm should concentrate on crawling clusters instead of travelling widely in the social network. For NeuroSearch different input types could be used to differentiate the neighbors thus forwarding only to a subset of neighbors. When a topic cluster is found by finding multiple matching results from peers close to each other, the other peers close to the topic cluster might gain more weight on the query decisions.

6.5.5.2 Resource discovery simulations

Current P2P research has concentrated on simulating resource discovery and topology management algorithms mainly in file-sharing P2P networks. However, in a social network the topology consists of friendship relations between the users and therefore the simulation scenario has different requirements. For example social network topology cannot be modified and therefore topology management is not needed. However, there is a need for topology awareness algorithm, which informs a peer about the structure of the topology near the peer. Also resource discovery algorithms have been designed to locate files anywhere in the network, but in a social network it is assumed that similar users tend to group with similar users (homophily) and therefore bring a structure of data to the network. This structure might be utilized by search algorithms.

The first simulation experiment aims at classifying which of the presented state-of-the-art P2P resource discovery algorithm information can be used in a social network. A representative set of different local information has been collected from literature and will be used for constructing a new social network search algorithm. This kind of information can be used:

- Hops (1,2,...) is the distance between querier and the queried node in the social network (nodes near the querier might be more socially relevant to the querier)
- ConnectivityRank (1,2,...) is the ranking of the queried node by the number of neighboring nodes of the queried node (highly connected nodes can reveal new nodes behind them better than low connected nodes)
- Reply% ([0,1]) is the amount of queries the node has replied divided by the total number of queries queried from the node (nodes which replied earlier to queries might be more likely to answer in the future)
- >RES (1,2,...) is the ranking of the queried node by the number of replies the node has returned [Yang & Garcia-Molina Improving Search in Peer-to-Peer Networks

2002] (nodes which replied earlier to queries might be more likely to answer in the future)

- BuddyListRank (1,2,...) is the rank of the queried node in the buddy list [Shao & Wang BuddyNet: History-Based P2P Search] (nodes which replied earlier to queries might be more likely to answer in the future)
- RepliesOnPath (0,1,2,...) is the number of nodes which have returned replies to this query on the shortest path between querier and the queried node in the social network (replying nodes on the shortest path can be an indication that this part of social network has matching replies)
- RepliesOnNeighbors (0,1,2,...) is the number of neighboring nodes which have returned replies to this query (nearby replying nodes can be an indication that this part of social network has matching replies)
- #NodesQueried (0,1,2,...) is the number of nodes currently evaluated for this query (gives an indication how far the querying has proceeded)
- #NodesFound (0,1,2,...) is the number of nodes which have replied to the current query (gives an indication how far the querying has proceeded)
- QueryPopularity ([0,1]) is #NodesFound/#NodesQueried (estimates the popularity of the current query in the social network)
- #QueriesFromReceiver (0,1,2,...) is the number of queries received earlier from the queried node (if a node receives queries from a certain node, then it is supposed that those nodes have common interests)
- QueriesFromReceiverRank (1,2,3,...) is the rank of #QueriesFromReceiver compared to other nodes evaluated for this query (#QueriesFromReceiver is not upper-bounded and grows when the number of queries grows, QueriesFromReceiverRank corrects the problem by scaling to the number of nodes in evaluation)
- Random% ([0,1]) is a random value between 0 and 1 (enables random walker type of algorithms)
- RandomRank (1,2,...) is a random rank value for each node (enables random walker type of algorithms)

These inputs will be fed to a linear classifier and weighted such that the output defines the rank of a peer compared to other candidate peers for the query.

The search scenario is also modified in such a way that each peer individually queries other peers and no forwarding is done in the network. This helps in overcoming the problems of trust related to forwarding of queries and motivation issues why mobile users would donate their battery life and bandwidth to unknown users. The downside is that the topology awareness algorithm needs to be designed light-weight enough to gather sufficient number of candidate peers for the queries.

As an output of the simulations we get a classification of inputs that suit well for social network search. We can restrict a minimal set of inputs or we can keep multiple inputs to increase the performance of the search algorithm. The evaluation will be done with the crawled social network data of Last.fm.

6.5.5.3 Prototype implementation

Prototype implementation is planned to be done using PAMP stack for Nokia phones. The idea is to provide a Google-type of search box to the user and display the matching query results from the social network. If the user is not happy with the results he can select to search more and thus explore larger area in the social network.

As an application scenario a distributed version of Last.fm functionalities can be used. The idea is to find in a distributed manner what other people are listening and thus get recommendations of new songs of similar taste. Listening could be enabled with a distributed search of songs and streaming of music from other mobile device. Search functionality can be used to find out blog entries containing discussions about a certain song/album/band etc. This kind of architecture removes a centralized Last.fm server and thus has no high-cost component in the system. Also the system is scalable, because now a huge amount of data can be made online to other users by storing the data on mobile devices. Similarly mobile device users can use their shared file folder to publish whatever kind of content and make it searchable for others. Possible candidates for indexing the shared folder contents could be Microsoft Search or Google Desktop indexing.

6.5.6 Summary

The advantages of distributed social P2P architecture in case of music community can be seen from the following table:

	Last.fm music site	*Distributed music community*
Hardware costs	High	Low
Bandwidth costs	High	Low
Total network traffic	Low, but has hot spots	High, but load-balanced
Index updates	Generates traffic	No traffic
Search complexity	Low	High
Search latency	Low	High
Search scope	Exhaustive	Limited query horizon
Storage space	High on server	Low
Fault-tolerance	Single point of failure	High availability in presence of failures

6.6 Conclusions and future work

We have presented three different approaches of building distributed peer-to-peer infrastructure in multiplatform environments. We have also shown required modifications and additions that have to be incorporated into platform's components in order to employ presented mechanism. In addition we described the instruments of active improvements in topologies of created networks. By the means of inter-platform discovery we give agents the opportunity to communicate, share services and resources beyond the boundaries of their home platforms.

Future work in WP6 will include incorporating one of the methods described in this report into the UBIWARE prototype. We also plan on conducting further research upon improving the efficiency of created network of agent platforms.

REFERENCES

Adamic, L.A. and Adar, E. (2003) Friends and neighbors on the Web, Social Networks, Vol. 25, No. 3, Elsevier, pp. 211-230.

Adamic, L.A. and Adar, E. (2004) Information flow in social groups, Physica A, Vol. 337, pp. 327-335.

Ahn, Y.-Y., Han, S., Kwak, H., Moon, S. and Jeong, H. (2007) Analysis of Topological Characteristics of Huge Online Social Networking Services. In Proceedings of the 16th International Conference on World Wide Web (WWW 2007), ACM, Banff, Alberta, Canada, pp. 835-844.

Anwar, Z., Yurcik, W., Pandey, V., Shankar A., Gupta, I. and Campbell, R.H. (2005) Leveraging Social-Network Infrastructure to Improve Peer-to-Peer Overlay Performance: Results from Orkut. ACM Computing Research Repository.

Bellifemine, F., Caire, G. and Greenwood, D. (2007) “Developing Multi-Agent Systems with JADE”, Wiley 2007.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) The semantic web (berners-lee et. al 2001). May 2001.

Bowring, E., Pearce, J.P., Portway, C., Jain, M. and Tambe, M. (2008) On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, pages 607-614, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

Cai, M. and Frank, M. (2004) Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *WWW ’04*: *Proceedings of the 13th international conference on World Wide Web*, pages 650–657, New York, NY, USA, 2004. ACM.

Chechetka, A., and Sycara, K. (2006) No-commitment branch and bound search for distributed constraint optimization. In AAMAS '06: Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems, pages 1427-1429, New York, NY, USA, 2006. ACM.

Cilibrasi, R., Vitanyi, P.(2007) The google similarity distance. IEEE Transactions on knowledge and data engineering 19(3), 370–383

Condie, T., Kamvar, S.D. and Garcia-Molina, H. (2004) Adaptive peer-to-peer topologies. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing* (P2P ’04), 2004

Curry, E., Chambers, D. and Lyons, G. (2003) “A JMS Message Transport Protocol for the JADE Platform” IEEE/WIC International Conference on Intelligent Agent Technology, 2003.

de Bruijn, J., Martin-Recuerda, F., Manov, D., Ehrig, M.(2004) D4.2.1 state-of-the-art-survey on ontology merging and aligning v1. SEKT Project deliverable D4.2.1

Dumais, S. (2003) Data-driven approaches to information access. Cognitive Science, 27(3), 491-524.

Ferber, J. (1999) *Multi-Agent Systems*: *An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Professional, February 1999.

FIPA (2004) “FIPA Agent Management Specification”, 2004.

Forbes (2006) http://www.forbes.com/intelligentinfrastructure/ 2006/04/27/video- youtube-myspace_cx_df_0428video.html (refereed: September 10, 2008).

Gershman, A., Meisels, A. and Zivan, R. (2006) Asynchronous forward-bounding for distributed constraints optimization. In Proc. 1st Intern. Workshop on Distributed and Speculative Constraint Processing, 2006.

Giunchiglia, F., Shvaiko, P.(2004) Semantic matching. The Knowledge Engineering Review 18:3 (Cambridge University Press) 265–280

Grinshpoun, T., Zazon, T., Binshtok, M. and Meisels, A. (2007) Termination problem of the APO algorithm, 2007.

Holvoet, T. and Weyns, D. (2005) On the role of environments in multiagent systems. In *E4MAS*, 2005.

Kalogeraki, V., Gunopulos, D. and Zeinalipour-Yazti, D. (2002) A Local Search Mechanism for Peer-to-Peer Networks. In Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02), ACM, McLean, Virginia, USA, pp. 300-307.

Kamvar, S., Schlosser, M. and Garcia-Molina, H. (2003) The Eigen- Trust Algorithm for Reputation Management in P2P Net- works. In WWW 2003, 2003

Landauer, T. K., Foltz, P. W., & Laham, D. (1998) Introduction to latent semantic analysis. Discourse Processes, 25, 259-284.

Langegger, A., Blochl, M., Woss, W. (2007) "Sharing Data on the Grid using Ontologies and distributed SPARQL Queries", *18th International Conference on Database and Expert Systems Applications, DEXA '07.* Regensburg, Germany, 3-7 Sept., 2007, pp.450-454.

Lemaire, B., Denhiére, G. (2004) Incremental construction of an associative network from a corpus. In K. D. Forbus, D. Gentner & T. Regier (Eds.), 26th Annual Meeting of the Cognitive Science Society, CogSci2004. Hillsdale, NJ: Lawrence Erlbaum Publisher.

Leskovec, J. and Horvitz, E. (2008) Planetary-Scale Views on a Large Instant-Messaging Network. In Proceeding of the 17th International Conference on World Wide Web (WWW 2008), ACM, Beijing, China, pp. 915-924.

Maheswaran, R.T., Pearce, J.P., and Tambe, M. (2004) Distributed algorithms for DCOP: A graphical-game-based approach. In PDCS, pages 432-439, 2004.

Mailler, R. and Lesser, V. (2004) Solving distributed constraint optimization problems using cooperative mediation. Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on, pages 438-445, 2004.

Meisels, A., Kaplansky, E., Razgon, I. and Zivan, R. (2002) Comparing performance of distributed constraints processing algorithms. In AAMAS, 2002.

Menascé, D. A. and Kanchanapalli, L. (2002) Probabilistic scalable P2P resource location services. ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2, pp. 48-58.

Mislove, A., Gummadi, K. P. and Druschel, P. (2006) Exploiting Social Networks for Internet Search. In Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets'06), Irvine, California, USA, pp. 79-84.

Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P. and Bhattacharjee, B. (2007) Measurement and Analysis of Online Social Networks. In Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, San Diego, California, USA, pp. 29-42.

Mitré, J. and Navarro-Moldes, L. (2004) P2P Architecture for Scientific Collaboration. In Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'04), pp. 95-100.

Modi, P.J., Shen, W., Tambe, M. and Yokoo, M. (2003) An asynchronous complete method for distributed constraint optimization. In AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pages 161-168, New York, NY, USA, 2003. ACM.

Modi, P.J., Tambe, M. and Yokoo, M. (2005) Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, 161:149-180, 2005.

Obermeier, P., Nixon, L. (2008) A Cost Model for Querying Distributed RDF Repositories, Advanced Reasoning on the Web workshop, European Semantic Web Conference (ESWC) 2008, Tenerife, Spain.

Okada, Y., Masui, K. and Kadobayashi, Y. (2005) Proposal of Social Internetworking. Human.Society@Internet 2005 (HSI 2005), pp. 114-124.

Omicini, A and Ossowski, S. (2003) Objective versus subjective coordination in the engineering of agent systems. In *AgentLink*, pages 179–202, 2003.

Pastor-Satorras, R. and Vespignani, A. (2004) Evolution and Structure of the Internet: A Statistical Physics Approach, Cambridge University Press.

Petcu, A. and Faltings, B. (2005) A Scalable Method for Multiagent Constraint Optimization. Technical report, 2005.

Pirolli, P. (2005). Rational analyses of information foraging on the Web. Cognitive Science, 29(3), 343-373.

Platon, E., Mamei, M., Sabouret, N., Honiden, N.and Parunak, H.V. (2007) Mechanisms for environments in multi-agent systems: Survey and opportunities. *Autonomous Agents and Multi-Agent Systems*, 14(1):31–47, 2007.

Platon, E., Sabouret, N. and Honiden, N. (2005) Oversensing with a softbody in the environment -another dimension of observation. In *Proceedings of Modelling Others from Observation'05*, 2005.

Quilitz, B., Leser, U. (2008) "Querying Distributed RDF Data Sources with SPARQL", *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008*, Tenerife, Canary Islands, Spain, June 1-5, 2008, pp.524-538.

Rahm, E., Bernstein, P.A. (2001) A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4)

Ramakrishnan, R. and Tomkins, A. (2007) Toward a PeopleWeb, IEEE Computer, Vol. 40, No. 8, pp. 63-72

Schelfthout, B and Leuven, D.K.U. (2005) Views: Customizable abstractions for context-aware applications in manets. In *In Workshop on Software Engineering for Large-Scale Multi-Agent Systems*. ACM Press, Digital Library, 2005.

Seaborne, A., Bizer, C. (2004) D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004).

Seaborne, A., Steer, D. and Williams, S. (2007) SQL-RDF. In W3C Workshop on RDF Access to Relational Databases, Cambridge (MA, USA), October 2007.

Shvaiko, P.(2004) A classification of schema-based matching approaches. In: Proceedings of the Meaning Coordination and Negotiation workshop at ISWC'04.

Shvaiko, P., Euzenat, J. (2005) A survey of schema-based matching approaches. Journal on Data Semantics IV,146–171

Singh, A. and Haahr, M. (2007) Decentralized Clustering In Pure P2P Overlay Networks Using Schelling's Model, IEEE International Conference on Communications (ICC '07), pp. 1860-1866.

Srinivasan, N. and Finin, T. (2002) "Enabling Peer to Peer SDP in Agents", Proc. 1st Int. Workshop on "Challenges in Open Agent Systems, July 2002, University of Bologna, held in conjunction with the 2002 Conf. on Autonomous Agents and Multiagent Systems

Tummolini, L., Castelfranchi, C., Ricci, A., Viroli, M., and Omicini, A. (2004) What i see is what you say: Coordination in a shared environment with behavioral implicit communication. In George A. Vouros, editor, *International Workshop on Coordination in Emergent Agent Societies*, pages 19–25. ECAI 2004, Valencia, Spain, 23.24. August, 2004.

Tummolini, L. et al. (2005) "exhibitionists" and "voyeurs" do it better: A shared environment for flexible coordination with tacit messages. In E4MAS, pages 215–231. Springer, 2005.

Varakantham, P., Nair, R., Tambe, M. and Yokoo, M. (2006) Winning back the CUP for distributed POMPDs: Planning over continuous belief spaces. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 289-296, New York, NY, USA, 2006. ACM.

Veksler, V. D., Gray, W. D. (2007) Mapping semantic relevancy of information displays. Paper presented at the CHI 2007, San Jose, CA.

Viroli, M. (2001) On observation as a coordination paradigm: An ontology and a formal framework. In *ACM Symposium on Applied Computing – Proceedings of 16th International Conference (SAC01)*, pages 166–175, Las Vegas (NV, pages 166–175. ACM, 2001.

Viroli, M. et al. (2007) Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):49–60, 2007.

Wellman, B. (1997) An Electronic Group is Virtually a Social Network. In Sara Kiesler (Ed.), Culture of the Internet, pp. 179-205, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T. and Ferber, J. (2004) Environments for multiagent systems state-of-the-art and research challenges. In *E4MAS*, pages 1–47. Springer, 2004.

Weyns, D., Steegmans, E. and Holvoet, T. (2003) Model for active perception in situated multi-agent systems. *Special Issue of Journal on Applied Artificial Intelligence*, 18:200–4, 2003.

Yeoh, W., Felner, A. and Koenig, S. (2008) BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 591-598, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

Yokoo, M. and Durfee, E.H. (1991) Distributed constraint optimization as a formal model of partially adversarial cooperation. Technical report, 1991.

Zhang, W., Wang, G., Xing, Z. and Wittenburg, L. (2005) Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artif. Intell., 161(1-2):55-87, 2005.

Appendix A: Section 1.1

Ontological Coordination in Multi-Agent Systems Built with S-APL

ABSTRACT

In open systems where the agents and resources composing them may be unknown at design time, or in dynamic systems evolving with time, there is a need to enable the agents to communicate their intentions with respect to future activities and resource utilization to resolve coordination issues dynamically. Ideally, we would like to allow ad-hoc interaction, where two stand-alone independently-designed systems are able to coordinate whenever a need arises. The Semantic Web based approach presented in this paper aims at enabling agents to coordinate without assuming any design-time ontological alignment of them. An agent can express an action intention using own vocabulary, and through the process of dynamic ontology linking other agents will be able to arrive at a practical interpretation of that intention. We also show how our approach can be realized on top of the Semantic Agent Programming Language.

1. INTRODUCTION

Coordination is one of the fundamental problems in systems composed of multiple interacting processes [15]. Coordination aims at avoiding negative interactions, e.g. when two processes conflict over the use of a non-shareable resource, as well as exploiting positive interactions, e.g. when an intermediate or final product of one process can be shared with another process to avoid unnecessary repetition of actions. A classic example of a negative interaction from the field of agent-based systems is two robots trying to pass thorough a door at the same time and blocking each other. A corresponding example of a positive interaction is a robot opening and closing the door when passing it while also letting the other robot to pass, in so saving it the need of opening/closing the door by itself.

The pre-dominant approach has been to hard-wire the coordination mechanism into the system structure [15]. Synchronization tools such as semaphores have been traditionally used to handle negative interactions, requiring every process to be programmed to check the semaphore before accessing the resource (like checking if there is an ”occupied” light over a lavatory door). If a resource is occupied by a process for a significant time, it would be clearly better for the other process to work on another its task rather than just wait. Under the traditional approach, realizing that as well as attempting to exploit any positive interactions is possible only through additional hard-wiring: the programs of the processes must have incorporated some knowledge about the behavior of each other.

This traditional approach becomes insufficient when considering more open systems, where the processes and resources composing the system may be unknown at design time [4]. In such systems, we ideally want computational processes to be able to reason about the coordination issues in their system, and resolve these issues autonomously [4]. One way to achieve this is to enable the relevant processes to *communicate their intentions* with respect to future activities and resource utilization [11]. Jennings [7] presents this as an issue of enabling individual agents to represent and reason about the actions, plans, and knowledge of other agents to coordinate with them.

Tamma at al. [15, 11] developed *an ontological framework* for dynamic coordination. They stated the need for an agreed common vocabulary, with a precise semantics, that is therefore suitable for representation as an ontology. [15] provided such an ontology that defined coordination in terms of agents carrying out activities involving some resources, which can be non-shareable, consumable, etc. [11] described then the rules for checking for conflicts among activities: e.g. if two activities overlap in time and require the same non-shareable resource, they are mutually-exclusive. [11] also described some possible coordination rules to be followed when a conflict of a certain type is detected.

The ontology of Tamma et al. is an *upper ontology*, i.e. an ontology which attempts to describe the concepts that are the same across all the domains of interest. Roughly speaking, the idea is to make the agents to communicate their intentions and actions using the upper-ontology concepts (i.e. ”resource”, ”activity”) rather than the domain-ontology concepts (e.g. ”printer”, ”printing document”) and in so to resolve the problem of evolving domains or domains not fully known at design time.

We build on this work of Tamma et al. We, however, observe a few drawbacks of the current solution:

- The traditional approach to coordination in a sense involves hard-wiring the domain ontology concepts into both agents that want to coordinate with each other. In the approach of Tamma et al., the upper ontology concepts are hard-wired into both instead. The latter is better than the former, yet still requires a design-phase alignment of agents and does not support for coordination with agents for which this was not done.

- Translating all coordination messages into the upper ontology may make them significantly longer. Also, when considering that in some cases the agents may actually share the domain ontology and in some other cases the receiver of the message may be familiar with a superclass of the unknown concept used in the message, bringing every conversation down to the upper ontology sounds somewhat unnatural.

On the other hand, we observe that the Semantic Web research explicitly addresses the possibility of multi-ontology systems. In open or evolving systems, different components would, in general, adopt different ontologies as either knowledge models of the environment or as knowledge models of own configuration, capabilities and behavior. Therefore, practical Semantic Web applications have to operate with heterogeneous data, which may be defined in terms of many different ontologies and may need to be combined, e.g., to answer specific queries [10]. At present, the standard technologies of the Semantic Web, such as RDF Schema (RDF-S) and Web Ontology Language (OWL), on the level of hierarchies of classes of entities, enable communications in which (we will discuss an example in Section 2):

- The sender of a message can express it in its own domain ontology and does not need to know any integrating upper ontology.
- Only the receiver of the message has to know the upper ontology and to have access to a formal definition of the domain ontology of the sender that links the concepts from that ontology to the upper ontology.

Therefore, an intelligent agent can potentially communicate with a ”stupid” agent (e.g. from a legacy system). It is even possible to connect two ”stupid” agents by putting an intelligent middleware in between.

Our approach to ontological coordination aims at enabling exactly the same: so that an agent can express its action intention according to its own domain ontology. Then, assuming that this ontology has a formal definition in terms of an upper ontology such as one by Tamma et al., the agent receiving the message will be able to interpret it and understand if there is any conflict or if there is a possibility to re-use any of the shareable results. In this paper, we describe this approach. In particular, we show how we realize it on top of the Semantic Agent Programming Language (S-APL) [9].

The rest of the paper is structured as follows. Section 2 presents our general framework and, then, Section 3 describes how the hierarchies of activity classes are modeled in S-APL. Section 4 briefly discusses the utilization of the basic definitions of activity classes in policies, e.g. for access control, while Section 5 describes how such definitions are further extended with coordination-related properties. Finally, Section 6 concludes the paper.

2. ONTOLOGICAL COORDINATION

Let us consider the following communication scenario, which is readily enabled by the standard technologies of the Semantic Web, namely RDF-S and OWL. Assume there are two agents; let us call one Enquirer and another Responder. Assume the Responder knows the following facts: *org:Mary rdf:type person:Woman* ; *person:hasSon org:Jack*, meaning that Mary is a woman and has a son Jack. (The syntax for RDF we use here is one of Turtle and of Notation3 [2]. We assume that the namespace *org*: refers to all entities related to an organization and *person*: denotes an ontology of people and relationships that is used in that organization).

Now assume that the Enquirer issues a SPARQL [17] query *SELECT ?x WHERE {?x rdf:type family:Mother}* (definition of prefixes is omitted), i.e. ”give me all entities that belong to the class *family:Mother*”. The result of executing this query is an empty set – the Responder does not have any facts that would directly match the pattern given. The Responder can, however, analyze the query and notice that the concept *family:Mother* is unknown to him. This can be done, e.g., by simply checking if he has any RDF triple involving this concept. So, the Responder decides to look for the ontology that defines it. In the simplest and common case, the definition of the prefix *family*: in the query will give the URL of the online OWL document defining the ontology in question. So, the Responder downloads it and obtains the information that *family:Mother* is a subclass of *human:Human* with the restriction that it must have a property *human:hasSex* with the value *human:FemaleSex* and must also have at least one property *human:hasChild*. (We assume that the namespace *human*: denotes some general upper ontology of humans). This additional information does not yet change the result of the query execution, because the Responder does not have a definition of his own *person*: ontology in terms of *human*: ontology. However, let us assume that he is able to locate (from a registry) and download such a definition. In so, the Responder obtains information that *person:Woman* is a subclass of *person:Person* which is in turn a subclass of *human:Human*, and that *person:Woman* has a restriction to have a property *human:hasSex* with the value *human:FemaleSex*. Also, he obtains the fact that *person:hasSon* is a sub-property of *human:hasChild*.

Then, the application of the standard RDF-S and OWL reasoning rules will infer that *org:Mary human:hasSex human:FemaleSex* (because she is known to be a woman) and also that *org:Mary human:hasChild org:Jack* (because having a son is a special case of having a child). Immediately, the OWL rules will conclude that *org:Mary rdf:type family:Mother* and this information will be sent back to the Enquirer. As can be seen, the concepts from the domain ontology used by the Enquirer were, through an upper ontology, dynamically linked to the concepts from the domain ontology used by the Responder. In so, the Enquirer was able to use his own concepts and, yet, the Responder was able to answer the question correctly.

Figure 1 depicts the logical components involved in this example. Our interpretation is that there are two upper ontologies here. One is the ontology of RDF-S/OWL itself – one operating with concepts such as class, subclass, property, restriction on property, etc. The other one is a basic agreed common vocabulary about human beings. Then, there should be a definition for each domain ontology involved – a definition that links the concepts from that ontology to an upper ontology (*human*: in this case), normally using concepts from another upper ontology (RDF-S/OWL properties in this case). Finally, at least one of the upper ontologies (RDF-S/OWL in this case) must come with a set of rules defined. These rules, when applied to the existing facts and domain ontologies' definitions, are supposed to infer new facts and in so to enable the understanding between agents.

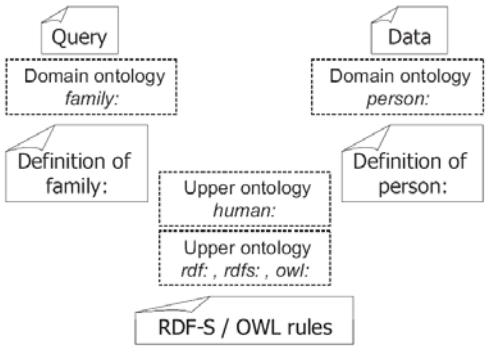


Figure 1: The example components

In the simple example above, the RDF graph we obtain after merging all sub-graphs in question is a sufficiently connected one, so that all the needed interpretations are directly possible. In many practical cases, this will not be a case, thus requiring *ontology alignment* (also known as ontology matching or ontology mapping). For example, we might not to have the fact that *person:hasSon* is a sub-property of *human:hasChild*. Ontology alignment is an important open challenge in the Semantic Web research (see e.g. [16]) and is outside the scope of this paper.

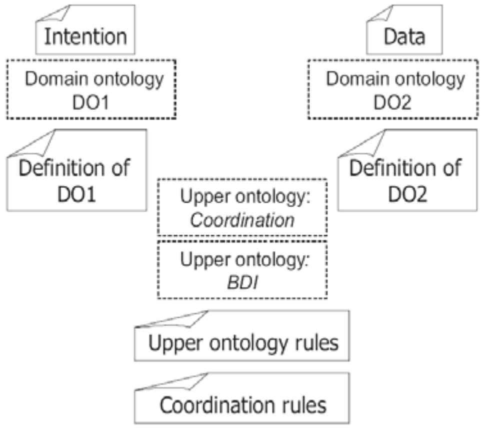


Figure 2: Ontological coordination framework

As was stated in Section 1, our goal is to enable more flexible and dynamic ontological coordination among agents, at least at the level of how RDF-S and OWL enable dynamic linking of entities' class hierarchies. By the analogy with Figure 1, Figure 2 depicts the logical components needed to realize that.

Let us assume that an agent communicates to another agent his intentions with respect to future actions, and let us assume that he does this using the concepts (activity names) from his own domain ontology unknown to the receiver. There are two upper ontologies involved. One is the *coordination ontology*, i.e. one that operates with the concepts such as activity and resource. In this paper, we assume the use of the ontology provided by Tamma et al. [15, 11]. The other upper ontology is the ontology of *mental attitudes* of agents. Since the Beliefs-Desires-Intentions (BDI) architecture [14] is quite a standard approach, Figure 2 assumes the BDI ontology in place of this ontology of mental attitudes. The definition of a domain ontology have to therefore link it to these two upper ontologies, in a way that will enable the *upper ontology rules* to do the following:

1. Interpret an expression of a mental attitude conveying an action intention to obtain the identifier of the intended activity.
2. Match the activity description in the domain ontology definition with the intention to understand what resources will be utilized (or results produced) by the intended action.

For example, in FIPA SL communication content language [6], the action intention is expressed using a construct like *(I (agent-identifier :name agent1) (done (action (agent-identifier :name agent1) (print some.pdf AgPS4e))))*. The upper ontology rules have to extract the name of the activity "print" and then, from the definition of that activity, understand that "AgPS4e" is the identifier of the resource (printer) that is going to be utilized by the intended action. As can be seen, these rules have to be tailored to a particular language used in communication. In our work, we utilize the Semantic Agent Programming Language (S-APL) [9] instead of SL or similar. An S-APL expression is an RDF graph itself, which simplifies describing activities in an ontology to enable the rules doing such interpretations (see Section 3). This also minimizes the need for tailoring.

Figure 2 also includes the *coordination rules* as the part of the framework. Those rules operate on the output of the upper ontology rules in order to e.g. identify conflicts between activities and propose resolution measures. In this paper, we simply assume the use of the coordination rules given in [11].

Assuming that an agent received a message and identified it as conveying an action intention of another agent, the flowchart of the ontology linking process is depicted in Figure 3. The terminator 'Done' implies only the end of this particular process. The upper ontology rules and the coordination rules can then trigger some follow-up actions.

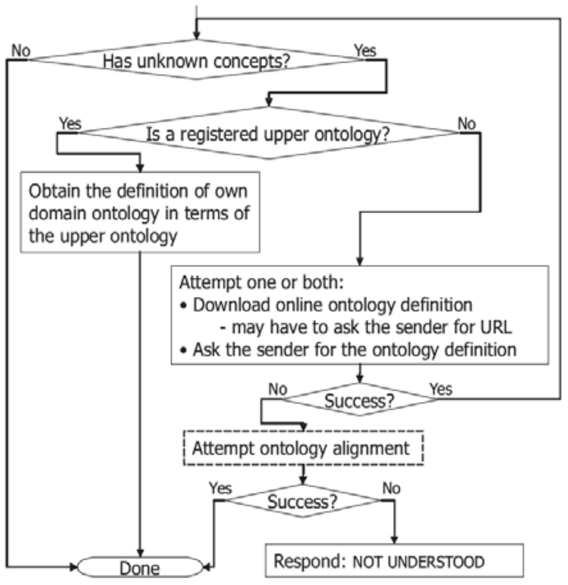

Has unknown concepts?
No
Yes
Is a registered upper ontology?
Yes
No
Obtain the definition of own domain ontology in terms of the upper ontology
Attempt one or both:
• Download online ontology definition
- may have to ask the sender for URL
• Ask the sender for the ontology definition
Success?
No
Yes
Attempt ontology alignment
Success?
Yes
No
Done
Respond: NOT UNDERSTOOD

Figure 3: Dynamic ontology linking process

3. DEFINING CLASSES OF ACTIVITIES

In this and the following sections, we show how the general ontological coordination framework described in Section 2 is realized with the Semantic Agent Programming Language (S-APL) [9] plus a set of new concepts we refer to as S-APL Schema (SAPL-S).

S-APL is an RDF-based language that integrates the semantic description of domain resources with the semantic prescription of the agents' behaviors. S-APL is a hybrid of semantic rule-based reasoning frameworks such as N3Logic [3] and agent programming languages (APLs) such as e.g. AgentSpeak(L) [13]. From the semantic reasoning point of view, S-APL is an extension of it with common APL features such as the BDI architecture, which implies an ability to describe goals and commitments – data items presence of which leads to some executable behavior, and an ability to link to sensors and actuators implemented in a procedural language, namely Java. From the APL point of view, S-APL is a language that has all the features (and more) of a common APL, while being RDF-based and thus providing advantages of semantic data model and reasoning. The syntax for RDF used in S-APL is one of Notation3 (N3) [2], which is more compact than RDF/XML. The implementation of the S-APL platform is built on the top of the Java Agent Development Framework (JADE) [1]. For an extensive presentation of S-APL see [9], and the technical details of the language can be found in [8].

In the context of this paper, we are mostly interested in one S-APL construct – *intention (commitment) to perform an action*. Such an intention is encoded in S-APL as:

{sapl:I sapl:do <action name>} sapl:configuredAs
{ <parameter> sapl:is <value>. ... }

sapl:I is an indicative resource that is assumed to be defined in the beliefs of an agent to be *owl*:sameAs the URI of that agent. Obviously, substituting *sapl*:I with an URI of another agent in the construct above would result in a description of somebody's else intention. A simple example of an intention to send a message to another agent follows. The "java:" namespace indicates that the action is a directly executable atomic behavior implemented in Java. Otherwise, the action would correspond to an S-APL plan (kind of subprogram) specified elsewhere. "p:" is the namespace for the parameters of standard (being a part of the S-APL platform) atomic behaviors.

```
{ sapl:I sapl:do
java:UBIWARE.MessageSenderBehavior}
   sapl:configuredAs {
     p:receiver sapl:is org:John.
     p:content sapl:is {org:Room1 org:temperature 25}
}
```

Note that one can easily put a construct specifying some other action intention as the contents of the message – in order to communicate that intention to the other agent.

An intention to perform an action, as any other S-APL construct, is just a logically connected set of RDF triples (Notation3 allows to have a compact representation but does not change the data model). If one wants to check if a larger S-APL dataset, e.g. the contents of a message, includes an intention to perform a particular action, one can simply run a query against the dataset. That query is given as a pattern, i.e. another set of RDF triples with some of the resources being variables. For instance, the pattern matching any own action intention is {*sapl:*I *sapl:do ?x} sapl:configuredAs ?y*. This is the same principle as followed in SPARQL for querying general RDF datasets.

Moreover, we can make the following observations:

- A pattern that is universally quantified by using variables, can be seen as the definition of a *class* of S-APL constructs, i.e. a class of agents' mental attitudes.
- When considering inheritance (class-subclass) hierarchies of mental attitudes, the definition of a subclass, in most cases, only introduces some additional restrictions on the variables used in the definition of the superclass. If, e.g. {*sapl:*I *sapl:do ?x} sapl:configuredAs ?y* is the definition of a general action, adding a statement *?*x *rdf:type org:PrintAction* may be used to create the definition of a class of printing actions.

In S-APL, it is easy to record such patterns as data, merge patterns when needed, and use patterns as queries against any given dataset – thus giving us all the needed means for modeling classes of agents' activities and utilizing them in rules.

S-APL Schema (namespace "sapls:" below) defines a set of concepts needed for such modeling. First, SAPL-S introduces a set of general classes of BDI mental attitudes, such as a goal or an action intention. SAPL-S ontology defines these classes using the statements of the type <*class*> *sapl:i*s <*pattern*>. Second, SAPL-S provides a property *sapls*:restriction that enables one to describe some additional restrictions on the pattern of a class to define some subclasses of it.

An action intention is defined in SAPL-S as:

```
sapls:Action sapl:is {
        {{?subject sapl:do ?behavior}
              sapl:configuredAs ?parameters} sapl:ID ?id
}
```

The wrapping with the property *sapl*:ID (see [9]) is included in order, when an action class definition is used as a query pattern, to receive the identifier of the matching action statement – to enable removing or modifying it if wished.

One can then define a subclass of *sapls*:Action, for example:

```
org:Scan rdfs:subClassOf sapls:Action;
    sapls:restriction {
            ?behavior rdf:type org:ScanAction.
            ?parameters sapl:hasMember
                        {org:device sapl:is ?device}.
            {?device rdf:type org:ScanDevice.
                        ?scanner sapl:expression ?device}
            sapl:or {?device org:hasPart ?scanner.
                        ?scanner rdf:type org:ScanDevice}
    }
```

This definition specifies that *org*:Scan is an action intention to perform an atomic behavior or a plan that is known to belong to the class *org*:ScanAction, and that has a parameter *org*:device referring to a device that either belongs to the class *org*:ScanDevice (a stand-alone scanner) or has a part that belongs to that class (a multi-function printer). This definition is made taking into account that we need to be able to specify which resource gets occupied by the activity in question. In this case, it is one whose URI will be bound to the variable ?scanner (note that sapl:expression as used above works as simple assignment). In Section 5, we will present the syntax for describing activities, including the resources they require.

Let us assume that we also define *org*:Print in exactly the same way as *org*:Scan, only with *org*:PrintAction and *org*:PrintDevice in places of *org*:ScanAction and *org*:ScanDevice, correspondingly. Then, we can also define *org*:Copy as intersection of both without additional restrictions:

```
org:Copy rdfs:subClassOf sapls:Scan, sapl:Print
```

Logically, the pattern defining a mental attitude class is obtained by merging its own *sapls*:restriction with all *sapls*:restriction of its super-classes and with *sapl*:is of the top of the hierarchy. Therefore, an activity is classified as *org*:Copy if it is implemented with a plan that is assigned to both *org*:ScanAction and *org*:PrintAction classes and that is performed on a device that has both an *org*:ScanDevice part and an *org*:PrintDevice part. Of course, the pattern will also match with a case where the whole device is tagged as both *org*:ScanDevice and *org*:PrintDevice. However, the latter would not be a good domain model since the scanning component and the printing component of a multi-function printer are normally independent resources, i.e., e.g., scanning a document does not block simultaneous printing of another document by another process.

The reason for separating the base part of a pattern given as *sapl*:is from the restrictions given as *sapls*:restriction is that the base part can be evaluated against any given dataset, e.g. the contents of a received message, while the restrictions are always evaluated against the general beliefs base of the agent.

4. ACTIVITY CLASSES IN POLICIES

Before continuing discussion of the main topic of this paper, namely dynamic coordination over shared resources and shareable results, let us briefly discuss the utilization of the basic definitions of activity classes in definitions of access control policies.

Semantic Web based approaches to access control policies have been developed in recent years [5, 12]. In both [5] and [12], the access control policies are defined in terms of *prohibition*s or *permission*s for certain actors to perform certain operations. Such policies may have a number of reasons behind them, with one of the reasons being coordination over shared resources. Such coordination is not dynamic, i.e. the conflicts are not resolved on per-instance basis. Rather, an agent with authority imposes some restriction on other agents' behaviors to avoid the conflicts as such. An example of such a policy could be "no employee other than the management is allowed to use company printers for copying". According to the syntax given in [5], such a policy could easily be defined by two statements ("rbac:" stands for role-based access control):

```
org:Employee rbac:prohibited org:Copy.
org:Management rbac:permitted org:Copy.
```

This definition assumes that *org*:Management is a subclass of *org*:Employee and that permissions have priority over prohibitions (this is not discussed in [5]), i.e. that the permission given to the management staff overrules the restriction put on a more general class of employees.

Combining policy definitions with definitions of the activity classes (Section 3) enables enforcement of the policies. An agent itself of an external supervisor can match the plans or intentions of the agent with the activity classes and then check if those are in the scopes of some defined policies. As a simplest reaction, an action that contradicts a policy can be blocked.

Dynamic ontology linking is also enabled. This means that a policy can be formulated using concepts originally unknown to the agent in question. For example, one may be informed about a prohibition to *org*:Copy while one may not know what *org*:Copy means. Yet, following the process sketched in Figure 3, one will be able to link this concept to *org*:Print and *org*:Scan and, if those are also unknown, link them to the upper S-APL BDI concepts.

In contrast to [5], [12] uses the concepts of prohibition and permission as the statement classes rather than predicates. The activity class is used as the predicate, and the policy statement is extended by specifying the class of the activity object. We utilize this approach in our work and represent an access control policy as in the example above in the form (”sbac:” stands for semantics-based access control):

```
{org:Employee org:Copy org:Printer}
                              sapl:is sbac:Prohibition.
{org:Management org:Copy org:Printer}
                              sapl:is sbac:Permission.
```

By substituting *org*:Printer with e.g. *org*:PrinterAg4, the policy can be modified into ”no employee other than the management is allowed to use for copying printers located on the 4th floor of the Agora building ”. Such policy is probably more realistic than the former because it may have a rationale that the managers use those printers for their higher-priority tasks and want to avoid possible delays.

In order to enable policy statements with objects, the definitions of *org*:Scan and of *org*:Print in Section 3 have to be extended with the statement ”?object sapl:expression ?device”, so that, after the matching an intention with the pattern, the variable ?object would be bound to the activity object. Note that the variable ?subject, which is need for both ways of defining policies, was already included in the definition of *sapls*:Action.

5. ANNOTATING ACTIVITIES FOR COORDINATION

In terms of Figure 2, the approach to defining activity classes described in Section 3 enables linking domain ontologies of activities to the upper BDI ontology and, therefore, the interpretation of expressed mental attitudes. The interpretation may give information about what activity is intended, by who (i.e. who is the subject), and on what object. As we discussed in Section 4, the ability of making such basic interpretations can already be utilized in policy mechanisms, such as those of access control. In order to enable more complex and dynamic coordination schemes, however, the definition of activities have to be also linked to the upper coordination ontology. As we already stated in Section 2, we use the coordination ontology given by Tamma at al. [15, 11]. In this section, we provide the syntax for the main concepts of that ontology and show how coordination-related properties are linked with basic definition of the activity classes. We use the namespace ”’coord:” to denote concepts belonging to the coordination ontology.

The set of properties used to describe activities follows:

- *<activity> coord:requires <variable>*. A resource utilized by the activity.
- *<activity> coord:shareableResult <variable>*. A result produced by the activity that is in principle shareable with another agent.
- *<activity> coord:earliestStartDate <variable or expression>*. The earliest time at which the activity may begin; null indicates that this information is not known. There are also similar predicates *coord:latestStartDate*, *coord:latestEndDate*, *coord:expectedDuration* as well as *coord:actualStartDate* and *coord:actualEndDate*.

It is uncommon (although in some cases possible) for a class of activities to have a defined resource URI, defined start time, etc. Therefore, we assume the objects of all the properties above to be variables which will be initialized when matching the class definition with an expressed action intention. For example, the *org:*Scan activity from Section 3 can be described with a statement:

```
org:Scan coord:requires ?scanner.
```

During the matching, the variable ?scanner will be given the URI of a stand-alone scanner or the scanning part of a multi-function printer. The statement above simply puts that this URI corresponds to a resource that is utilized by the activity.

We could also define a subclass of *org*:Scan, *org*:ScanToFile, which allows saving the result of scanning into a file whose name is given as the parameter *org*:saveTo, and then add a description that this file is shareable with other agents:

```
org:ScanToFile rdfs:subClassOf org:Scan;
    sapls:restriction {
         ?parameters sapl:hasMember
                              {org:saveTo sapl:is ?file}.
    };
    coord:shareableResult ?file.
```

Similarly, if the parameters of the action intention include the timestamp when the action is planned to be executed, one could use a variable receiving this timestamp when annotating the activity class with time-related properties. Here, arithmetic expressions are allowed, e.g. ”?time+1000” (milliseconds). Our present solution does not provide for including a query that would connect the properties of an activity input with time-related estimates. For example, the expected duration of the printing activity is related to the number of pages to print. Of course, realizing this is possible by including needed statements (like *?file org:hasPages ?number. ?duration sapl:expression ”?number*1000”*) into the activity class definition. This is not, however, a proper modeling because absence of information about the number of pages of the printed document would lead to not counting the action as printing, while it should only lead to inability to provide the duration estimate. At present, we are working on a general approach that will enable linking the object of a statement to a query; that approach will resolve also the specific issue above while keeping the syntax compact.

Given such annotations of activity classes, the interpretation rules are to have the basic form as follows:

```
{...
 ?x rdfs:subClassOf sapls:Action.
 sapls:Action sapl:is ?base.
 ?x sapls:restriction ?restriction.
 ?dataset sapl:hasMember {?base sapl:is sapl:true}.
 ?restriction sapl:is sapl:true.
 {?x coord:requires ?res.
  ?resource sapl:expression "valueOf(?res)"
 } sapl:is sapl:Optional.
 ...
} => {
 _:?id rdf:type coord:Activity; rdf:type ?x;
       coord:actor ?subject;
       coord:requires ?resource ...
}
```

Here, for the sake of brevity, we assume that there exist additional rules that do the pre-processing of the activity class hierarchies. These rules extend *sapls*:restriction of an activity class with *sapls*:restriction of its superclasses and also extend the activity class annotation with *coord*:requires, *coord*:sharedResult, etc. of the superclasses. (It is also possible, of course, to write a longer interpretation rule that does not require such pre-processing). The variable ?dataset is assumed to refer to the dataset which is being searched for an action intention, e.g. the contents of a message. *sapl*:Optional wraps a non-mandatory part of the query, similarly to a corresponding construct in SPARQL. If the variable ?resource will not get bound, the statement in the right hand of the rule that uses this variable will not be created. In this example, the activity URI is generated as the blank node prefix " :" plus the identifier of the intention statement.

When two agents attempt to utilize the same resource, the type and the effect of the conflict depends on the resource. The set of important subclasses of the class *coord:*Resource follow:

- *coord:ShareableResource*. The resource that can be simultaneously used by two activities, e.g. a computing unit. Simultaneous use normally results in activities impeding, but not blocking, each other.
- *coord:NonShareableResource*. The resource that can only be used by one activity at a time.
- *coord:ConsumableResource*. A special type of a non-shareable resource that is also consumed when used, i.e. not available for any other activity afterwards.
- *coord:CloneableResource*. The resource that can be duplicated for use in several activities, e.g. an electronic document.

An example of a coordination rule follows:

```
{?x rdf:type coord:Activity; coord:actor sapl:I;
    coord:requires ?r.
 ?y rdf:type coord:Activity; coord:actor ?agent;
    coord:requires ?r.
 ?r rdf:type coord:NonShareableResource.
 ?ca rdf:type coord:ContractualAuthority;
     coord:hasSourceAgent ?agent;
     coord:hasTargetAgent sapl:I
} => {... postpone or suspend own activity ...}
```

This example assumes the use of the concept of the *operationa*l *relationshi*p from the coordination ontology in [11]. An operation relationship is a relationship between agents that implies the priority of one over the other. In [11], ContractualAuthority is a subclass of OperationalRelationship that implies that the "source" agent has the priority over the "target" agent. The operational relationship concept is important for coordination, we believe, however, that it should be a part of a larger organizational ontology rather than embedded into the coordination ontology.

6. CONCLUSIONS

When considering systems where the agents and resources composing them may be unknown at design time, or systems evolving with time, there is a need to enable the agents to communicate their intentions with respect to future activities and resource utilization and to resolve coordination issues at run-time. In an ideal case, we would like also to allow ad-hoc interaction of systems, where two stand-alone independently-designed systems are able to communicate and coordinate whenever a need arises. Consider, for example, two robots with totally unrelated goals who need to coordinate their activities when they happen to work in the same physical space.

The Semantic Web based approach presented in this paper aims at enabling agents to coordinate without assuming any design-time ontological alignment of them. An agent can express an action intention using own vocabulary, and through the process of dynamic ontology linking other agents will be able to arrive at a practical interpretation of that intention. The definition of the domain ontology in terms of an upper ontology must be provided. However, such a definition is external to the agents and may be added later, when an agent is already in the operation.

In result, an intelligent agent can potentially communicate with a "stupid" agent, e.g. from a legacy system. It is also possible to connect two "stupid" agents by putting an intelligent middleware in between. This work has been performed in a research project where the latter case is a major motivation. The interests of the project industrial partners are in Enterprise Application Integration and data integration, with an accent on enabling new intelligent business processes in systems created by interconnecting independently-designed applications and data sources that often do not share a common data model or even ontology.

Some important challenges to be addressed in the future work follow. One challenge is that the current coordination ontology of Tamma et al. does not provide for explicit modeling of the effect of activities on the resources they utilize. The ontology allows resources to be *consumed* by activities, but this is modeled by a resource property (see Section 5) rather than by an activity property. Therefore, there is no way of distinguishing between activities that consume the resource, e.g. printing on paper, and activities that reserve the resource (make unavailable to other activities) without the consumption, e.g. transporting a package of paper from one place to another. Similarly, in many cases, it is needed to distinguish between an activity that destroys a resource (e.g. erases a file) and an activity that uses it (e.g. reads the file). Additionally, one may want to be able to distinguish between consuming/destroying a resource and changing it. For example, printing on a sheet of paper does not destroy it. It consumes it in the sense that it makes the sheet unavailable for future printing activities; however the sheet remains usable for other activities that do not depend on the sheet being clean. In short, the coordination ontology and the corresponding modeling syntax has to be extended with constructs that will enable describing the effect of an activity on a resource or an attribute of that resource.

Another challenge is related to increasing the flexibility of the approach by allowing some of an activity's parameters to come from the background knowledge of the listener agent rather than from the message. For example, an agent X can inform an agent Y about the intention to print a document without specifying the printer. Yet, Y could know what printer X normally uses and make the interpretation based on that. An even more interesting scenario is where X informs Y about an intention to ask some third agent, Z (e.g. a secretary), to print a document for him.

The final challenge we list is related to the need for more complex coordination rules. The rules discussed in [11] treat conflicts that are identifiable from the activities' descriptions alone. However, if an activity changes an attribute of a resource, the resource may undergo some follow-up changes due to environmental causes, thus leading to a conflict. For example, the activity of opening a food container would not be seen as conflicting with a later activity of consuming the food in the container, unless considering that the food in an open container will spoil faster than in a closed one. This implies that for many practical cases the identification of conflicts has to be performed as reasoning or planning process rather than based on straightforward rules.

7. REFERENCES

[1] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[2] T. Berners-Lee. *Notation 3: A readable language for data on the Web*. Online: http://www.w3.org/DesignIssues/Notation3.html.

[3] T. Berners-Lee, D. Connoly, L. Kagal, Y. Scharf, and J. Hendler. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3):249–269, 2008.

[4] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *Proc. 1st Intl. Conf. on Multi-Agent Systems*, pages 73–80. AAAI Press, 1995.

[5] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. ROWLBAC: Role based access control in OWL. In *Proc. ACM Symposium on Access Control Models and Technologies*, pages 73–82, 2008.

[6] Foundation for Intelligent Physical Agents. *FIPA SL Content Language Specification*. Online: http://www.fipa.org/specs/fipa00008/SC00008I.pdf.

[7] N. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[8] A. Katasonov. *UBIWARE Platform and Semantic Agent Programming Language (S-APL). Developer's guide*, 2008. Online: http://users.jyu.fi/~akataso/SAPLguide.pdf.

[9] A. Katasonov and V. Terziyan. Semantic agent programming language (S-APL): A middleware platform for the Semantic web. In *Proc. 2nd IEEE International Conference on Semantic Computing*, pages 504–511, 2008.

[10] E. Motta and M. Sabou. Next generation semantic web applications. In *Proc. ACM Asian Semantic Web Conference, LNCS vol.4185*, pages 24–29, 2006.

[11] T. Moyaux, B. Lithgow-Smith, S. Paurobally, V. Tamma, and M. Wooldridge. Towards service-oriented ontology-based coordination. In *Proc. IEEE Intl. Conference on Web Services*, pages 265–274, 2006.

[12] A. Naumenko. Semantics-based access control – Ontologies and feasibility study of policy enforcement function. In *Proc. ACM 3rd International Conference on Web Information Systems and Technologies, Volume Internet Technologies*, pages 150–155, 2007.

[13] A. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNCS vol.1038*, pages 42–55. Springer, 1996.

[14] A. Rao and M. Georgeff. Modeling rational agents within a BDI architecture. In *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, 1991.

[15] V. Tamma, C. Aart, T. Moyaux, S. Paurobally, B. Lithgow-Smith, and M. Wooldridge. An ontological framework for dynamic coordination. In *Proc. 4th Semantic Web Conference, LNCS vol. 3729*, pages 638–652. Springer, 2005.

[16] V. Tamma and T. Payne. Is a Semantic web agent a knowledge-savvy agent? *IEEE Intelligent Systems*, 23(4):82–85, 2008.

[17] W3C. *SPARQL Query Language for RDF. W3C Recommendation 15 January 2008*. Online: http://www.w3.org/TR/rdf-sparql-query/.

Appendix B: UBIWARE Publications List

Katasonov A. and Terziyan V. (2009) Ontological Coordination in Multi-Agent Systems Built with S-APL, submitted 14.10.2008 to the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2009)

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 (in press)

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

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.

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, pp. 169-178

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, 2008, SERSC publisher, ISSN: 1975-4094, 18 pp.

Katasonov, A., Terziyan, V., SmartResource Platform and Semantic Agent Programming Language (S-APL), In: P. Petta et al. (Eds.), *Proceedings of the 5-th German Conference on Multi-Agent System Technologies (MATES'07)***,** 24-26 September, 2007, Leipzig, Germany, Springer, LNAI 4687 pp. 25-36.

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.

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.

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.

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.

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.

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.

Naumenko A., Semantics-Based Access Control in Business Networks, Jyvaskyla Studies in Computing, *PhD Thesis*, Volume 78, Jyvaskyla University Printing House, 215 pages, 2007.

Srirama, S., and Naumenko, A., (2007). 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.

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.

Naumenko A., Katasonov A., Terziyan V., A Security Framework for Smart Ubiquitous Industrial Resources, In: R. Gonzalves, J.P. Müller, 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.

Katasonov A., Kaykova O., Khriyenko O., Loboda O., Naumenko A., Nikitin S., Terziyan V., The Central Principles and Tools of UBIWARE, *Technical Report (Deliverable D 1.1)*, UBIWARE Tekes Project, Agora Center, University of Jyvaskyla, May-October 2007, 118 pp.