We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

8

Using Semantic Technology to Enable Behavioural Coordination of Heterogeneous Systems

Artem Katasonov
VTT Technical Research Centre Finland
Vagan Terziyan
University of Jyväskylä Finland

1. Introduction

Coordination is one of the fundamental problems in systems composed of multiple interacting processes (Tamma et al., 2005). 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. On the Web, the coordination has not yet been treated much as traditional Web applications and services are normally isolated from each other, run on separate computing systems and, therefore, do not do not have other types of interaction beyond *using* each other. However, as the Internet grows towards the Internet of Things, where the physical and digital worlds will be interconnected, where e.g. Web services will control various physical processes, the problem of coordination becomes more and more critical also in the Web domain.

The predominant approach to coordination has been to hard-wire the coordination mechanism into the system structure (Tamma et al., 2005). 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 behaviours 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 (Decker & Lesser, 1995). 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 (Decker & Lesser, 1995). We would like to even allow ad-hoc interaction, where two stand-alone independently-designed systems are able to coordinate whenever a need arises. One way towards achieving this is to enable the relevant processes to *communicate their intentions* with respect to future activities and resource utilization (Moyaux et al., 2006). Jennings at al. (1998) present 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. In other words, there is a need for the interacting processes, e.g. software agents, Web services, etc, to be able to communicate not only about the external world, i.e. the domain, but also about their own abilities, goals, as well as the current and intended actions.

In the case of highly heterogeneous systems, enabling such a dynamic coordination among them is an even harder problem than more traditional problems of data-level or protocol-level heterogeneity. Tamma and colleagues (Tamma et al., 2005; Moyaux et al., 2006) 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. Tamma et al. (2005) provided such an ontology that defined coordination in terms of *agents* carrying out *activities* involving some *resources*, which can be non-shareable, consumable, etc. Moyaux et al. (2006) described then the rules for checking for conflicts among activities: e.g. if two activities overlap in time and require the same resource that is known to be non-shareable, they are mutually-exclusive. They 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 and colleagues. 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 ontological 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 super-class 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 behaviour. 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 (Motta & Sabou, 2006). 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 entities' classes, 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.

We would like to disclaim that we do not imply here the use of any automated ontology mapping (also known as ontology matching and ontology alignment, see Tamma & Payne, 2008; Shvaiko & Euzenatsh, 2008), which is an imprecise, e.g. statistical, process of identifying relationships between concepts in two domain ontologies. We speak of a case where the concepts from both domain ontologies were manually, by human designers, linked to a single upper ontology. Then, the needed automatic process consists only of locating, accessing and use of relevant ontology specifications. This process we refer to in this chapter as *ontology linking*. The definition of a domain ontology in terms of an upper ontology acts as an annotation, i.e., is external to the agents and therefore may be added when an agent is already in the operation. 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 with its own actions or if there is a possibility to re-use any of the shareable results. In this chapter, we describe this approach. In particular, we show how we realize it with the *Semantic Agent Programming Language (S-APL)* (Katasonov & Terziyan, 2008).

2. Ontological coordination principles

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, see Berners-Lee, 2000a. 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 (W3C, 2008) 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 for describing human beings.) 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 (e.g. 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 when formulating a question and, yet, the Responder was able to answer the question correctly.

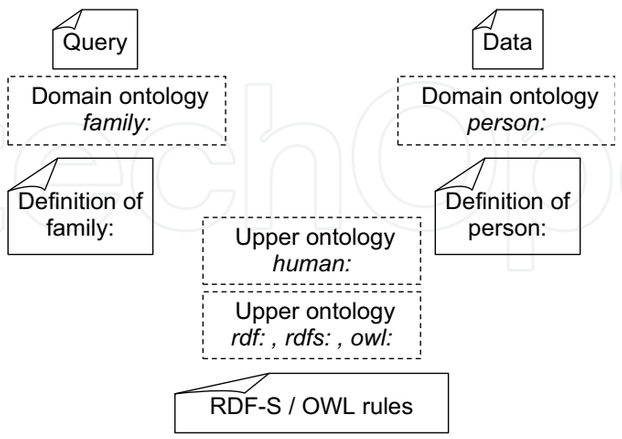

Fig. 1. The logical components of the example

Figure 1 depicts the logical components involved in this example. There are two upper ontologies involved. 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 specification 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.
In the simple example above, the RDF graph we obtain after merging all sub-graphs in question (data plus two ontology specifications) 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 know 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. Tamma & Payne, 2008; Shvaiko & Euzenatsh, 2008) and is outside the scope of this chapter. Note that we include "attempt ontology alignment" in Figure 3 below as the last resort for a case when ontology linking did not succeed; we do not discuss, however, how this can be done.

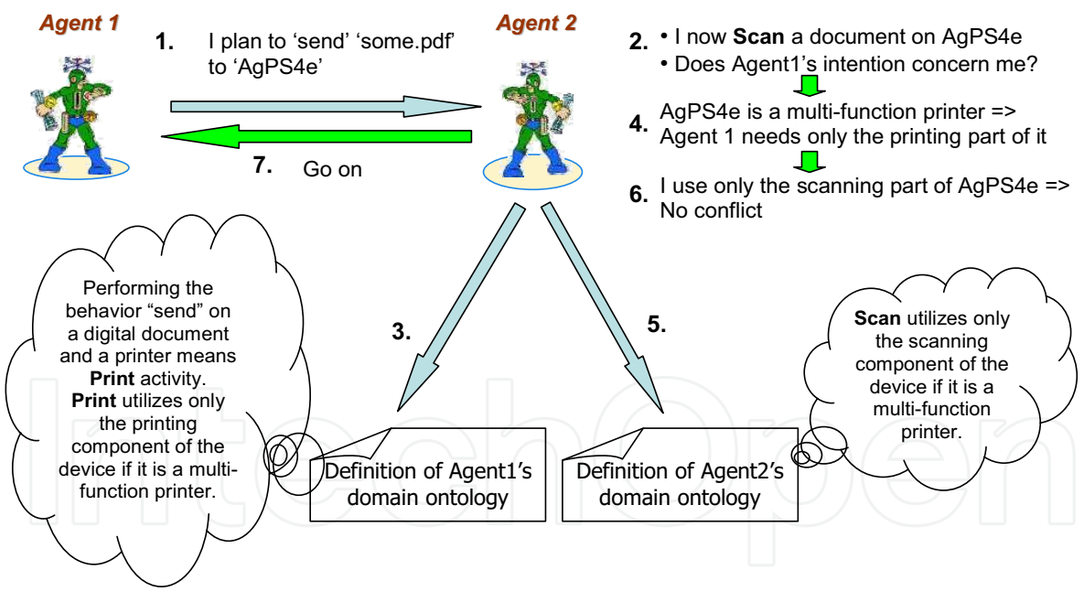

Fig. 2. Ontological coordination situation

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. Figure 2 depicts an example situation. Agent1 informs Agent2 that he plans to "send" some resource called "some.pdf" to some resource called "AgPS4e". Let us assume that Agent2 can recognize the former resource as a digital

document and the latter resource as a multi-function printer. Agent2 is currently utilizing (or plans to) the scanning function of AgPS4e. So, an obvious question appears is there any conflict between this activity and Agent1's intention.
Following a similar workflow as in RDF-S/OWL example earlier, Agent2 could try to locate a specification of the domain ontology of activities used by Agent1. From such a specification, he would receive information that, in the vocabulary of Agent1, sending a document to a printer corresponds to executing *Print* activity, that *Print* holds the printer for the whole time of the activity, and that if a multi-function printer is in question (that can also scan, copy, etc.) *Print* requires only the printing component of it. Then, if this has not been done yet, Agent2 would have to locate the definition of his own domain ontology of activities to obtain similar information about his own ongoing activity *Scan*. Finally, combining all this information, Agent2 would infer that he and Agent1 need different components of the multi-function printer AgPS4e that can be engaged independently and, therefore, there is no conflict.

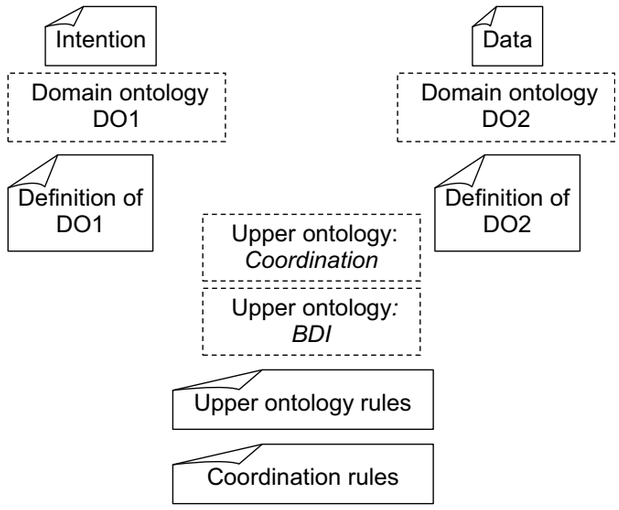

Fig. 3. Ontological coordination framework

By the analogy with Figure 1, Figure 3 depicts the logical components needed to realize this. 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 a vocabulary 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, like one provided by Tamma and colleagues (Tamma et al., 2005; Moyaux et al., 2006). The other upper ontology is the ontology of *mental attitudes* of agents. Since the Beliefs-Desires-Intentions (BDI) architecture (Rao & Georgeff, 1991) is quite a standard approach, Figure 3 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 and results produced by the intended action.

For example, in FIPA SL communication content language (FIPA, 2002), an action intention is expressed using a construct like *(I (agent-identifier :name agent1) (done (action (agent-identifier :name agent1) (print some.pdf AgPS4e))))*. In a simplest case, the upper ontology rules have to extract the name of the activity "print" and then, from the semantic 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, as well as corresponding activities' definitions, have to be tailored to a particular language used in communication. In addition, more complex cases are likely and have to be handled, where the activity name as modelled in the ontology is not present directly in the expressed intention but has to be inferred from the action parameters. As in the example depicted in Figure 2, the intention could have been given as "send some.pdf AgPS4e". Then, the fact that the printing activity is meant has to be inferred from combining a more general and ambiguous "send" with known classes of the resources some.pdf (a document) and AgPS4e (a printer).

In our work, we utilize the Semantic Agent Programming Language (S-APL) (Katasonov & Terziyan, 2008) instead of SL or similar. An S-APL expression is an RDF graph itself, which greatly simplifies describing activities in an ontology to enable the rules to match them with expressed intentions and to do all needed interpretations (see Section 4).

Figure 3 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, like those described in Moyaux et al. (2006).

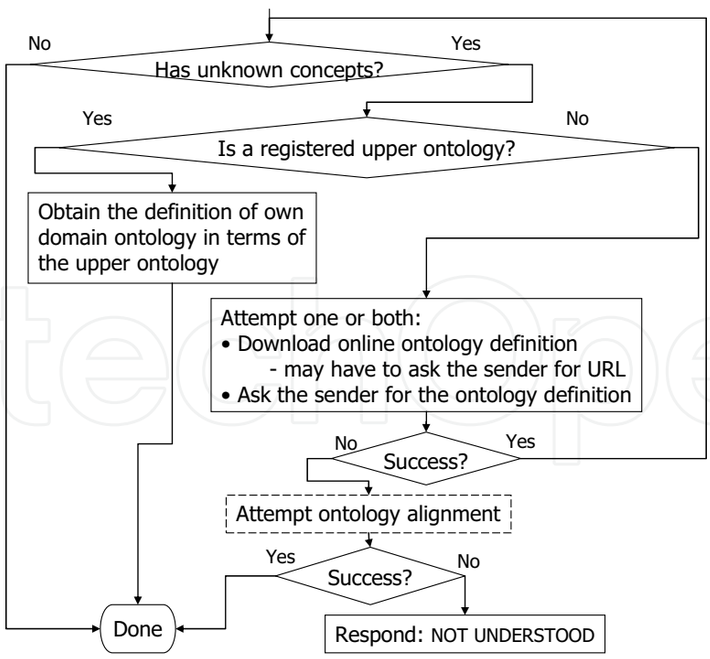

Fig. 4. Dynamic ontology linking process

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 4. 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.

3. Semantic Agent Programming Language (S-APL)

The main motivation for the development of the Semantic Agent Programming Language (S-APL) (Katasonov & Terziyan, 2008) was to facilitate the dynamic coordination of heterogeneous systems according to the principles presented in Section 2. S-APL provides a common medium for realizing all the stages of the ontological coordination framework described.

S-APL is an RDF-based language that integrates the semantic description of domain resources with the semantic prescription of the agents' behaviours. S-APL is a hybrid of semantic rule-based reasoning frameworks such as N3Logic (Berners-Lee et al., 2008) and agent programming languages (APLs) such as e.g. AgentSpeak(L) (Rao, 1996). From the semantic reasoning point of view, S-APL is an extension of CWM (Berners-Lee, 200b) 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 behaviour, 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. S-APL can be used as a programming language as well as the content language in the inter-agent communications: in querying for data, in requesting for action, as well as in communicating plans and intentions.

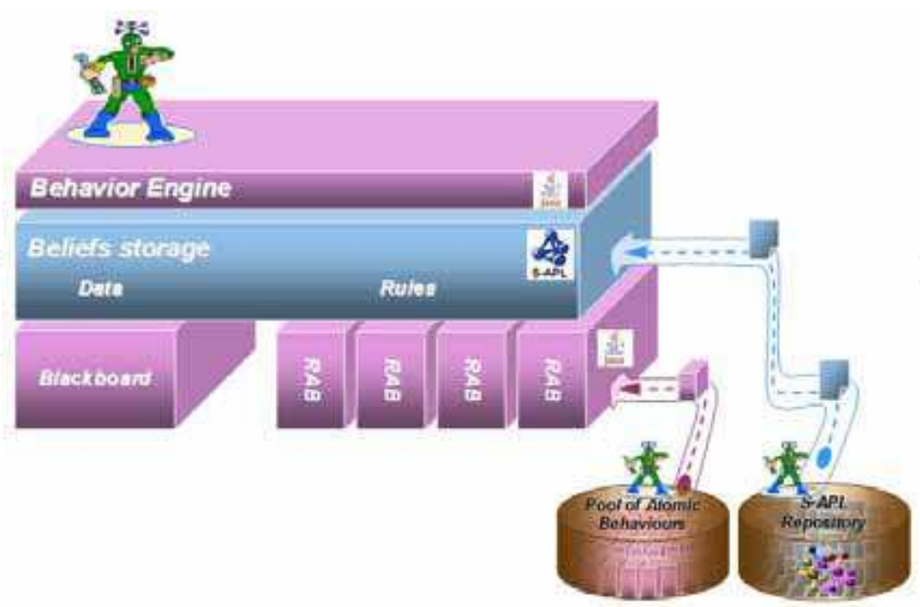

Fig. 5. Architecture of an S-APL agent

The architecture of an S-APL agent is depicted in Figure 5. The basic 3-layer structure is common for the APL approach. There is the behaviour engine implemented in Java, a declarative middle-layer, and a set of sensors and actuators which are again Java components. The latter we refer to as *Reusable Atomic Behaviours (RABs)*. We do not restrict RABs to be *only* sensors or actuators, i.e. components concerned with the agent's environment. A RAB can also be a reasoner (data-processor) if some of the logic needed is impossible or is not efficient to realize with S-APL, or if one wants to enable an agent to do some other kind of reasoning beyond the rule-based one. We also equip each agent with a blackboard, through which RABs can exchange arbitrary Java objects, for cases where the use of semantic data is not possible.

The middle layer is the agent' beliefs storage. What differentiates S-APL from traditional APLs is that S-APL is RDF-based. In addition to the advantages of the semantic data model and reasoning, an extra advantage is that in S-APL the difference between the data and the program code is only logical but not any principal. Data and code use the same storage, not two separate ones. This means that a rule upon its execution can add or remove another rule, the existence or absence of a rule can be used as a premise of another rule, and so on. None of these is normally possible in traditional APLs treating rules as special data structures principally different from normal beliefs which are n-ary predicates. S-APL is very symmetric with respect to this – anything that can be done to a simple RDF statement can also be done to any belief structure of any complexity.

As Figure 5 stresses, an S-APL agent can obtain the needed data and rules not only from local or online documents, but also through querying S-APL repositories. Such a repository, for example, can be maintained by some organization and include prescriptions (lists of duties) corresponding to the organizational roles that the agents are supposed to play. In our implementation, such querying is performed as inter-agent action with FIPA ACL messaging but does not involve any query or content languages beyond S-APL itself. As can be seen from Figure 5, agents also can load RABs remotely. This is done as an exception mechanism triggered when a rule prescribes engaging a RAB while the agent does not have it available. Thus, organizations are able to provide not only the rules to follow but also the tools needed for that.

Our implementation of the S-APL platform is built on the top of the Java Agent Development Framework (JADE) (Bellifemine et al., 2007). JADE provides communication infrastructure, agent lifecycle management, agent directory-based discovery and other standard services.

The syntax for RDF used in S-APL is one of Notation3 (N3) (Berners-Lee, 2000a) and S-APL utilizes the syntax for rules very similar to that of N3Logic (Berners-Lee et al., 2008). N3 was proposed as a more compact, better readable and more expressive alternative to the dominant notation for RDF, which is RDF/XML. One special feature of N3 is the concept of formula that allows RDF graphs to be quoted within RDF graphs, e.g. *{org:room1 org:hasTemperature 25} org:measuredBy org:sensor1*. An important convention is that a statement inside a formula is not considered as asserted, i.e., as a general truth. In a sense, it is a truth only inside a context defined by the statement about the formula and the outer formulas. In S-APL, we refer to formulae as *context containers*. The top level of the S-APL beliefs storage, i.e. what is the general truth for the agent, we refer to as *general context* or just *G*.

The technical details of S-APL can be found in Katasonov (2008). Below, we describe the main constructs of S-APL. We use three namespaces: *sapl:* for S-APL constructs, *java:* for

RABs, and *p:* for the parameters of standard (being a part of the S-APL platform) atomic behaviours. The namespace *org:* is used for resources that are assumed to be defined elsewhere.
The two constructs below are equivalent and define a simple belief. The latter is introduced for syntactic reasons.

```
org:room1 org:hasTemperature 25.
{org:room1 org:hasTemperature 25} sapl:is sapl:true.
```

The next two constructs add context information:

```
{org:room1 org:hasTemperature 25} org:measuredBy org:sensor1.
{org:room1 org:hasTemperature 25} sapl:is sapl:true;
                                  org:measuredBy org:sensor1.
```

The former states that "sensor1 measured the temperature to be 25" without stating that "the agent believes that the temperature is 25". In contrast, the latter states both. This demonstrates a specific convention of S-APL: rather than doing several statements about one container, "{...} P1 O1; P2 O2" leads to linking the statements inside the formula to two separate containers. Then, using *sapl*:true it is also possible to link some statements to a container and to one of its nested containers.
The goals of the agent and the things that the agent believes to be false (not just unknown) are defined, correspondingly, as:

```
sapl:I sapl:want {org:room1 org:hasTemperature 25}.
{org:room1 org:hasTemperature 25} sapl:is sapl:false.
```

sapl:I is an indicative resource that is defined inside the beliefs of an agent to be *owl*:sameAs the URI of that agent. A specific convention of S-APL is that e.g. "sapl:I sapl:want {A B C}. sapl:I sapl:want {D E F}" is the same as "sapl:I sapl:want {A B C. D E F}". In other words, the context containers are joined if they are defined through statements with the same two non-container resources.
The commitment to an action is specified as follows:

```
{sapl:I sapl:do java:ubiware.shared.MessageSenderBehavior}
     sapl:configuredAs {
          p:receiver sapl:is org:John.
          p:content sapl:is {org:room1 org:hasTemperature 25}.
          sapl:Success sapl:add
               {org:John sapl:is org:notified}
     }
```

The *java:* namespace indicates that the action is a RAB. Otherwise, the action would correspond to an S-APL plan (kind of subprogram) specified elsewhere. When the behaviour engine finds such a belief in G, it executes the RAB and removes the commitment. In the configuration part, one may use special statements to add or remove beliefs. The subject can be *sapl*:Start, *sapl*:End, *sapl*:Success, *sapl*:Fail. The predicate is either *sapl*:add or *sapl*:remove. Using such statements, one can also easily specify sequential plans: *{sapl:I sapl:do ...} sapl:configuredAs {... sapl:Success sapl:add {{sapl:I sapl:do ...} sapl:configuresAs {...}}}.*
Beliefs can also be added or removed through explicit mental actions:

```
sapl:I sapl:remove {?x sapl:is org:notified}
```

```
sapl:I sapl:add {org:John sapl:is org:notified}
```

sapl:remove uses its object as a pattern that is matched with G and removes all beliefs that match. Use of variables (as ?x above), filtering conditions, etc. is possible. *sapl*:add adds its object to G. It does not need to be normally used, since just stating something is the same as adding it to G. This construct is only needed when one wants to postpone the creating of the belief until the stage of the agent run-time cycle iteration when commitments are treated, or when one uses as the object a variable holding the ID of a statement or a container (see below).
The conditional commitment is specified as:

```
{
      {?room org:hasTemperature ?temp} org:measuredBy ?sensor.
      ?temp > 30
} => {...}
```

=> and > are shorthands for *sapl*:implies and *sapl*:gt, correspondingly. The object of *sapl*:gt and other filtering predicates (>=, <, <=, =, !=) is an expression that can utilize arithmetic operations, functions like *abs*, *floor*, *random*, etc. and string-processing functions like *length*, *startsWith*, *substring*, etc. When the behaviour engine finds in G a belief as above and finds out that all the conditions in the subject context container are met, it copies to G all the beliefs from the object container substituting variables with their values. Those can be simple beliefs and/or commitments, unconditional or conditional. S-APL allows a variable value to substitute a part of a resource, e.g. "logs/?today/received". Such a liberty is in contrast with, e.g., N3Logic approach where a variable value can only be a substitute for the whole resource; however, it was shown to greatly simplify the programming.
As with any commitments, the conditional commitment is removed after successful execution. In order to create a persistent rule, the => statement has to be wrapped as:

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

A specific convention of S-APL is that if there are several possible solutions to the query in the left side of =>, the right side is copied by default for the first-found solution only. One can use *sapl*:All wrappings to define that the right part has to be copied several times: for every unique value of some variable of every unique combination of the values of some variables. These wrapping can be used in either the left or the right side:

```
{ {{ ... } sapl:All ?x} sapl:All ?y } => {...}
{...} => { {{...} sapl:All ?x} sapl:All ?y }
```

sapl:All on the right side is allowed to enable defining different wrappings for different (top-level) resulting statements, e.g. *{...} => {X Y Z . {{?x L ?y} sapl:All ?y. A B ?x} sapl:All ?x}*. On the left side of =>, *sapl*:All must always wrap the whole contents of the container.
Other solutions set modifiers are also available, namely *sapl*:OrderBy, *sapl*:OrderByDescending, *sapl*:Limit, and *sapl*:Offset. The meaning of those are the same as of their equivalents in SPARQL. One can also wrap a condition in the left side of => with *sapl*:Optional to have the same effect as SPARQL's OPTIONAL, and connect two conditions with *sapl*:or to have the same effect as SPARQL's UNION. It is also possible to specify exclusive conditions, i.e. ones that must not be know to be true, by using the wrapping *sapl*:I *sapl*:doNotBelieve {...}.
There are also several of alternatives to =>, including:

```
{...} -> {...} ; sapl:else {...}
{...} ==> {...}
```

-> and ==> are the shorthands for *sapl*:impliesNow and *sapl*:infers. -> specifies a conditional action rather than a commitment: it is checked only once and removed even if it was false. One can also combine it with *sapl*:else to specify the beliefs that have to be added if the condition was false. ==> works almost the same as => with the following difference. If one uses => inside a persistent rule for semantic inference (generating new facts from existing ones), one needs to: (1) add to the head of the rule the negation of the tail the rule – to avoid continuous non-stop execution of the rule; (2) use a set of *sapl*:All wrappings – for all relevant variables – to enforce that the rule infers all possible facts in one iteration. When using ==>, these two things are done automatically – negation of the tail is checked and the rule is executed for every solution found.
One can also define new calculated variables:

```
{?person org:hasHeight ?h. ?feet sapl:expression ''?h/0.3048''.
     ?m sapl:min ?feet } => {...}
```

sapl:expression gives to the new variable the value coming from evaluating an expression. *sapl*:min is a special predicate operating on the set of matching solutions rather than on a particular solution – it determines the minimum value of the variable. The other predicates from the same group are *sapl*:max, *sapl*:sum, *sapl*:count (number of groups when grouped by values of one or several variables) and *sapl*:countGroupedBy (number of members in each group).
Variable can also refer to IDs of statements and context containers, and one can use the predicates *sapl*:hasMember, *sapl*:memberOf, *rdf*:subject, *rdf*:predicate and *rdf*:object. After obtaining the ID of the container with *?x org:accordingTo org:Bill*, one can do the following things:

```
{... {?x sapl:is sapl:true} org:accordingTo org:John} => {...}
?x sapl:is sapl:true
sapl:I sapl:add ?x
sapl:I sapl:remove ?x
sapl:I sapl:erase ?x
?x sapl:hasMember {org:room1 org:hasTemperature 25}
```

The first construct defines a query that is evaluated as true iff any belief that is found in the context container ?x has a match in the context container {} org:accordingTo org:John. The second one links the statements from ?x to G, while the third copies them to G. The fourth uses the contents of ?x as the pattern for removing beliefs from G, while the fifth erases the container ?x itself. Finally, the sixth adds to the container ?x a new statement.
There are several ways to create a variable holding IDs of some statements:

```
{{?room org:hasTemperature 25} sapl:ID ?x}
                                org:accordingTo org:Bill
{?x rdf:predicate org:hasTemperature} org:accordingTo org:Bill
{?x sapl:is sapl:true} org:accordingTo org:Bill
?c org:accordingTo org:Bill. ?c sapl:hasMember ?x
```

The first will find all the statements inside the container {} org:accordingTo org:Bill that match the pattern given, while the second all the statements with the predicate org:hasTemperature. The third and the fourth will find all the statements in that container.
One can then use a query like *{{?x sapl:is sapl:true} org:accordingTo org:Bill. {?x sapl:is sapl:true} org:accordingTo org:John} => {...}*, which is evaluated as true if there is at least one belief from the first container that has a match in the second container. One can also use *sapl*:true, *sapl*:add, *sapl*:remove, *sapl*:erase, *sapl*:hasMember and *sapl*:memberOf to do the same things as listed above for containers, but for a single statement.

4. 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) (see Section 3) plus a set of additional concepts we refer to as S-APL Schema (SAPL-S).
In this context, we are mostly interested in one S-APL construct – *intention (commitment) to perform an action*. As described in Section 3, such an intention is encoded in S-APL as:

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

Such a construct, when found in the agent's beliefs, leads to execution of the specified action. *sapl*:I is an indicative resource that is 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 was provided in Section 3. Note that one can easily put a construct specifying another action intention as the contents of the message (in place of the single triple in that example) – 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. First, 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. Second, 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 super-class. 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 modelling classes of agents' activities and utilizing them in rules.

S-APL Schema (namespace *sapls:* below) defines a set of concepts needed for such modelling. 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:is* <*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 is included in order to, when an action class definition is used as a query pattern, 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 behaviour 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 6, 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, *org*:PrintDevice and ?printer in places of *org*:ScanAction, *org*:ScanDevice and ?scanner 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.

5. Using activity classes in policies

Before continuing discussion of the main topic of this chapter, 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 (Finin et al., 2008; Naumenko, 2007). In both Finin et al. (2008) and Naumenko (2007), the access control policies are defined in terms of *prohibitions* or *permissions* 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' behaviours 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 Finin et al. (2008), 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 Finin et al., 2008), 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 Finin et al. (2008), Naumenko (2007) 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 such 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 needed for both ways of defining policies, was already included in the definition of *sapls:*Action.

6. Annotating activities for coordination

In terms of Figure 2, the approach to defining activity classes described in Section 4 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*.
In this section, we describe such an ontology and show how coordination-related properties are linked to basic definitions of the activity classes as presented in Section 4. We use the namespace *coord:* to denote concepts belonging to this ontology. As was mentioned in Section 1, with respect to a coordination ontology, we build on the work of Tamma and colleagues (Tamma et al., 2005; Moyaux et al., 2006). Below, we first describe concepts that correspond directly to those of Tamma et al. After that, we comment on limitations of the ontology of Tamma et al. and present few extensions to it.
The central concepts of the coordination ontology are:

- *coord:Agent* – a thing that able of performing some actions that may require coordination.
- *coord:Process* – something that that changes the state of the environment in some way.
- *coord:NonCoordinableActivity* – a subclass of coord:Process for which coordination is not possible. Non-coordinable activities can be natural events or other processes that are outside of control of the agents comprising the system in question.
- *coord:CoordinableActivity* – a process performed by an agent that is a part of the system in question, which therefore can be coordinated.
- *coord:Resource* – something that may be required to expedite an activity.

When two activities require the same resource, the type and the effect of the interaction depends on what resource is in question. The set of important subclasses of the class *coord*:Resource are (note that Tamma et al. model these as boolean properties of *coord*:Resource rather than subclasses of it):

- *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.

The set of properties used to describe activities follows:

- *coord:actor* – the agent performing the activity
- *coord:requires* – a resource utilized by the activity.
- *coord:shareableResult* – a result produced by the activity that is in principle shareable with another activities.
- *coord:earliestStartDate* – 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*.
- *coord:status* – the execution status of the activity, which can be one of the following: requested, proposed, scheduled, continuing, suspended, failed, succeeded.

These properties have a double use: in operational data to describe the instances of *coord*:CoordinableActivity and in activity ontologies to describe subclasses of *sapls*:Action. The former use is straightforward, e.g:

```
_:act397 rdf:type coord:CoordinableActivity;
          coord:requires org:AgPS4e
```

On the other hand, it would be uncommon for an ontological description of an activity class to have a defined resource URI (i.e. always the same resource), defined start time, etc. Therefore, in this use, the objects of all the properties above are allowed to be variables which are to be initialized when matching the class definition with an expressed action intention. For example, the *org*:Scan activity from Section 4 can be described with a statement:

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

As commented earlier, 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 activities and 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* (in milliseconds).
Given such annotations of activity classes, the interpretation rules in S-APL 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:CoordinableActivity; 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 super-classes and also extend the activity class annotation with *coord*:requires, *coord*:sharedResult, etc. of the super-classes. (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.
One limitation of the ontology of Tamma et al. is that it does not provide for explicit modelling of the effect of activities on the resources they utilize. The ontology includes a possibility to define a resource as being *consumable* (see above). However, 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 has to be extended with constructs that will enable describing the effect of an activity on a resource or an attribute of that resource.

There is also a challenge 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 expressed action intention directly. 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. In addition, some of the resources used by an activity might not be mentioned in the action intention. For example, a printing intention would not normally mention paper in the expressed parameters. Yet, we may want to be able to specify that the paper will be consumed in the activity. The ontology of Tamma et al. does not include concepts or properties to enable this.

Finally, we may want to be able to connect the parameters of an action intention with time-related estimates. For example, the expected duration of the printing activity is related to the number of pages to print. Realizing this is possible by including an extra statement like *?file org:hasPages ?pages* into the activity class definition, and then by annotating the activity class as <*activity*> *coord:expectedDuration "?pages*1000"*. We can also wrap this extra statement with {} *sapl*:is *sapl*:Optional, so that absence of information about the number of pages of the printed document would not lead to not counting the action as printing, but only to inability to provide the duration estimate. However, we believe that the definition of an activity class and its coordination-related annotation should not be mixed in such a way. Rather, a separate construct is needed.

For these reasons above, we extended the coordination ontology with the following properties that are to be used in activity ontologies to describe subclasses of *sapls*:Action:

- *coord:assumption* – a pattern for querying the beliefs storage of the agent to extend the information given explicitly in the action intention. In principle, this construct is very similar to the concept of *precondition* of an activity. However, the statements given are not treated as required since we can not assume the agent to be omniscient.
- *coord:effect* – the expected rational effect of the activity. Specifies changes to the resources that the activity uses or other environment entities.

The definition below of a subclass of *org*:Print, *org*:PrintFile, provides an example of using these two properties:

```
org:PrintFile rdfs:subClassOf org:Print;
      sapls:restriction {
            ?parameters sapl:hasMember
                  {org:input sapl:is ?file}.
      };
      coord:assumption {
            ?file org:hasPages ?pages.
            ?printer org:hasSheetsOfPaper ?sheets.
            ?remain sapl:expression ''?sheets-?pages''
      };
      coord:expectedDuration "?pages*1000";
      coord:effect {?printer org:hasSheetsOfPaper ?remain}
```

The last element in the ontological coordination framework presented in Section 2 and depicted in Figure 3 is the coordination rules. Those rules attempt to identify conflicts between activities and propose resolution measures. An example of a coordination rule in S-APL 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 org:ContractualAuthority;
          org:hasSourceAgent ?agent;
          org:hasTargetAgent sapl:I
} => {... postpone or suspend own activity ...}
```

This example uses the concept of the *operational relationship* from Moyaux et al. (2006). An operation relationship is a relationship between agents that implies the priority of one over the other. ContractualAuthority is a subclass of OperationalRelationship that implies that the "source" agent has the priority over the "target" agent. Moyaux et al. (2006) put these concepts as part of the coordination ontology. The operational relationship concept is important for coordination, however, we believe that it should be a part of a larger organizational ontology rather than embedded into the coordination ontology. This is why in the example above we did not put these concepts into the *coord:* namespace.

7. 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.

Enabling such a dynamic coordination among highly heterogeneous applications is an even harder problem than more traditional problems of data-level or protocol-level heterogeneity. While the Semantic Web technologies are designed to handle the latter problems, they also provide a basis for handling the coordination problem.

The Semantic Web based approach presented in this chapter 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 UBIWARE (Katasonov et al., 2008) 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.

In this chapter, we first described our general framework for dynamic ontological coordination. Then, we showed how we realize this framework on top of the Semantic Agent Programming Language. In so, this chapter provided a functional vertical solution. One can develop agents with S-APL and instruct them to communicate their intentions using S-APL as the communication content language, i.e. basically send to other agents small pieces of their own code. Then, one can develop needed definitions of the ontologies of activities (Section 4), extend them with coordination-related properties (Section 6) and implement various coordination rules (Section 6), thus getting a fully working solution. Additionally, one can specify and enforce access control policies (Section 5). Of course, the value of the general framework goes beyond this particular S-APL implementation.

One limitation of our present approach, which poses an important challenge to be addressed in the future work, is the following. We assumed so far that the conflicts among activities 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.

8. References

Bellifemine, F. L., Caire, G. & Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*, Wiley

Berners-Lee, T. (2000a) *Notation 3: A readable language for data on the Web*, online: http://www.w3.org/DesignIssues/Notation3.html

Berners-Lee, T. (2000b) *CWM: A general-purpose data processor for the semantic web*, online: http://www.w3.org/2000/10/swap/doc/cwm

Berners-Lee, T., Connoly, D., Kagal, L., Scharf, Y. & Hendler J. (2008). N3Logic: A logical framework for the World Wide Web, *Theory and Practice of Logic Programming*, Vol.8, No.3, pp. 249–269

Decker, K. & Lesser, V. (1995) Designing a family of coordination algorithms. *Proceedings of 1st Intl. Conf. on Multi-Agent Systems*, pp. 73–80. AAAI Press

Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W. & Thuraisingham, B. (2008). ROWLBAC: Role based access control in OWL, *Proceedings of ACM Symposium on Access Control Models and Technologies*, pp. 73–82

Foundation for Intelligent Physical Agents (2002). *FIPA SL Content Language Specification*, online: http://www.fipa.org/specs/fipa00008/SC00008I.pdf

Jennings, N., Sycara, K. P. & Wooldridge, M. (1998). A roadmap of agent research and development, *Autonomous Agents and Multi-Agent Systems*, Vol. 1, No.1, pp.7–38

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

Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S. & Terziyan, V. (2008). Smart semantic middleware for the internet of things, *Proceedings of 5th International Conference on Informatics in Control, Automation and Robotics, Vol. ICSO*, pp. 169–178

Katasonov A. & Terziyan, V. (2008) Semantic agent programming language (S-APL): A middleware platform for the Semantic Web, *Proceedings of 2nd IEEE International Conference on Semantic Computing*, pp. 504–511

Motta, E. & Sabou, M. (2006). Next generation semantic web applications, *Proceedings of ACM Asian Semantic Web Conference, LNCS vol.4185*, pp. 24–29

Moyaux, T., Lithgow-Smith, B., Paurobally, S., Tamma, V. & Wooldridge, M. (2006). Towards service-oriented ontology-based coordination, *Proceedings of IEEE International Conference on Web Services*, pp. 265–274

Naumenko, A. (2007). Semantics-based access control – Ontologies and feasibility study of policy enforcement function, *Proceedings of 3rd ACM International Conference on Web Information Systems and Technologies, Vol. Internet Technologies*, pp. 150–155

Rao, A. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language, *Proceddings of 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNCS vol.1038*, pp. 42–55

Rao, A. & Georgeff, M. (1991). Modeling rational agents within a BDI architecture, *Proceedings of 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 473–484

Shvaiko, P. & Euzenat, J. (2008). Ten challenges for ontology matching, *Proceedings of 7th Conference on Ontologies, Databases, and Applications of Semantics*

Tamma, V., Aart, C., Moyaux, T., Paurobally, S., Lithgow-Smith, B. & Wooldridge, M. (2005). An ontological framework for dynamic coordination, *Proceedings of 4th Semantic Web Conference, LNCS vol. 3729*, pp. 638–652

Tamma, V. & Payne, T. (2008). Is a Semantic web agent a knowledge-savvy agent? *IEEE Intelligent Systems*, Vol. 23, No.4, pp. 82–85

W3C (2008). *SPARQL Query Language for RDF*, W3C Recommendation 15 January 2008, online: http://www.w3.org/TR/rdf-sparql-query/

Semantic Web
Edited by Gang Wu

ISBN 978-953-7619-54-1
Hard cover, 310 pages
Publisher InTech
Published online 01, January, 2010
Published in print edition January, 2010

Having lived with the World Wide Web for twenty years, surfing the Web becomes a way of our life that cannot be separated. From latest news, photo sharing, social activities, to research collaborations and even commercial activities and government affairs, almost all kinds of information are available and processible via the Web. While people are appreciating the great invention, the father of the Web, Sir Tim Berners-Lee, has started the plan for the next generation of the Web, the Semantic Web. Unlike the Web that was originally designed for reading, the Semantic Web aims at a more intelligent Web severing machines as well as people. The idea behind it is simple: machines can automatically process or “understand” the information, if explicit meanings are given to it. In this way, it facilitates sharing and reuse of data across applications, enterprises, and communities. According to the organisation of the book, the intended readers may come from two groups, i.e. those whose interests include Semantic Web and want to catch on the state-of-the-art research progress in this field; and those who urgently need or just intend to seek help from the Semantic Web. In this sense, readers are not limited to the computer science. Everyone is welcome to find their possible intersection of the Semantic Web.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Artem Katasonov and Vagan Terziyan (2010). Using Semantic Technology to Enable Behavioural Coordination of Heterogeneous Systems, Semantic Web, Gang Wu (Ed.), ISBN: 978-953-7619-54-1, InTech, Available from: http://www.intechopen.com/books/semantic-web/using-semantic-technology-to-enable-behavioural-coordination-of-heterogeneous-systems

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen