Abstract

In open systems where the components, i.e. the agents and the resources, may be unknown at design time, or in dynamic and self-organizing 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 [17]. 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 through 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 [17]. 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 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 [12]. 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 et al. [17, 12] 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. [17] provided such an ontology that defined coordination in terms of agents carrying out activities involving some resources, which can be non-shareable, consumable, etc. [12] 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. [12] 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
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 [11]. 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.

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 [18, 16]), 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 paper 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 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) [10].

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 and gives future work directions.

## 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 [19] 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 human: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 2. Ontological coordination framework](image)

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. [17, 12]. The other upper ontology is the ontology of mental attitudes of agents. Since the Beliefs-Desires-Intentions (BDI) architecture [15] 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

\[
(I \text{ (agent-identifier :name agent1)} \text{ (done (action (agent-identifier :name agent1) (print some.pdf AgPS4e))))}.
\]

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) [10] 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 [12].

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.

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) [10] 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) [14]. 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 [10], and the technical details of the language can be found in [8]. S-APL is assumed to be used both as the programming language and a communication content language.

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:

\[
\{\text{sapl:\text{I}} \text{ sapl:\text{do <action name>}} \text{ sapl:\text{configuredAs}} \{ \text{<parameter>} \text{ sapl:is <value>. ... } \}\}
\]
Such a construct, when found in the agent’s beliefs, leads to execution of the specified action. \( \text{sapl} : \text{I} \) is an indicative resource that is assumed to be defined in the beliefs of an agent to be \( \text{owl:sameAs} \) the URI of that agent. Obviously, substituting \( \text{sapl} : \text{I} \) 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.

\[
\begin{align*}
\{ \text{sapl} : \text{I} & \text{ sapl} : \text{do java:} \text{ubiware.MessageSenderBehavior} \\
\text{sapl} : \text{configuredAs} & \{ \\
\text{p:receiver} & \text{ sapl} : \text{is org:John} \\
\text{p:content} & \text{ sapl} : \text{is org:Room1 org:temperature 25} \\
\} \\
\end{align*}
\]

Note that one can easily put a construct specifying some other action intention as the contents of the message (in place of the single triple in the example above) – 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 (Notation\(3\) 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 \( \{ \text{sapl} : \text{I} \text{ sapl} : \text{do } ?x \text{ sapl} : \text{configuredAs } ?y \}. \) 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 \textit{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 superclass. If, e.g. \( \{ \text{sapl} : \text{I} \text{ sapl} : \text{do } ?x \text{ sapl} : \text{configuredAs } ?y \} \) is the definition of a general action, adding a statement \( ?x \text{ rdf:type org:PrintAction} \) 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 ”\text{sapls}”:’ below) defines a set of concepts needed for such modeling. First, S-APL-S introduces a set of general classes of BDI mental attitudes, such as a goal or an action intention. S-APL-S ontology defines these classes using the statements of the type \(<\text{class}> \text{sapls:is } \text{patterndeclaration}>\). Second, S-APL-S provides a property \text{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 S-APL-S as:

\[
\begin{align*}
\text{sapls:Action} & \text{ sapl:is } \{ \\
\{ ?\text{subject} \text{ sapl:do } ?\text{behavior} \} & \text{ sapl:configuredAs } \{ ?\text{parameters} \} \text{ sapl:ID } ?\text{id} \\
\} \\
\text{The wrapping with the property } \text{sapl:ID} \text{ (see [10]) is included in order, when an action class definition is used as a query pattern, to receive the identifier of the matching action statement – to enable removing or modifying it if wished. One can then define a subclass of } \text{sapls:Action}, \text{ for example:} \\
\text{org:Scan rdfs:subClassOf sapls:Action; } \\
\text{sapls:restriction } \{ \\
\text{?behavior rdf:type org:ScanAction. } \\
\text{?parameters sapl:hasMember } \\
\text{\{org:device sapl:is } ?\text{device}. \\
\text{?device rdf:type org:ScanDevice. } \\
\text{?scanner sapl:expression } ?\text{device}. \\
\text{?scanner rdf:type org:ScanDevice} \\
\} \\
\}
\end{align*}
\]

This definition specifies that \text{org:Scan} is an action intention to perform an atomic behavior or a plan that is known to belong to the class \text{org:ScanAction}, and that has a parameter \text{org:device} referring to a device that either belongs to the class \text{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 \text{?scanner} (note that \text{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 \text{org:Print} in exactly the same way as \text{org:Scan}, only with \text{org:PrintAction} and \text{org:PrintDevice} in places of \text{org:ScanAction} and \text{org:ScanDevice}, correspondingly. Then, we can also define \text{org:Copy} as intersection of both without additional restrictions:

\[
\text{org:Copy rdfs:subClassOf sapls:Scan, sapl:Print} \\
\]

Logically, the pattern defining a mental attitude class is obtained by merging its own \text{sapls:restriction} with all
sapls:restriction of its super-classes and with sapls: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 sapls: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, 13]. In both [5] and [13], 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’ 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], [13] 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}
sapls:is sbac:Prohibition.
{org:Management org:Copy org:Printer}
sapls: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 sapls: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. [17, 12]. 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}.
  };
```

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.
  ?dataset sapl:hasMember { ?base sapl:is true}.
  ?restriction sapl:is true.
  ...}
```

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:

```
?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 *operational relationship* from the coordination ontology in [12]. An operation relationship is a relationship between agents that implies the priority of one over the other. In [12], 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 UBIWARE [9] 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 paper, 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 paper 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 3), extend them with coordination-related properties (Section 5) and implement various coordination rules (Section 5), thus getting a fully working solution. Additionally, one can specify and enforce access control policies (Section 4). Of course, the value of the general framework goes beyond this particular S-APL implementation.

Some limitations of our present approach are described below. These limitations present important challenges to be addressed in the future work. 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 [12] 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.

Acknowledgments

This work is performed in UBIWARE project, which is supported by Tekes (Finnish National Agency for Technology and Innovation) and industrial partners Metso, Fingrid, Inno-W, Nokia, and ABB.

References