Enabling real world Semantic Web applications through a coordination middleware

Robert Tolksdorf, Lyndon Nixon, Elena Paslaru Bontas, Duc Minh Nguyen, Franziska Liebsch
tolk, nixon, paslaru, nguyen@inf.fu-berlin.de, franziska@adestiny.de

Free University of Berlin
Institute for Computer Science
AG Networked Information Systems
Takustr. 9, D-14195 Berlin, Germany
http://nbi.inf.fu-berlin.de

Abstract. In a real world scenario Semantic Web applications must be capable to cope with the large scale, distributed, heterogeneous, unreliable and insecure environment of the World Wide Web if they are to truly represent added value to Web users. This includes issues of persistent storage, efficient reasoning, data mediation, scalability, distribution of data, fault tolerance and security. In this paper we present a coordination middleware for the Semantic Web and demonstrate its relevance to these vital issues for Semantic Web applications by elaborating a typical use case from the traffic management domain.

1 Introduction

The Semantic Web research effort is focused on providing suitable knowledge representation models and techniques for the large scale distributed environment of the Web. However there is less consideration for the particular requirements of applications which will be implemented to work with these models and techniques in order to provide intelligent Semantic Web-based functionality to users. Such applications must be equally capable to cope with the large scale, distributed, heterogeneous, unreliable and insecure environment of the World Wide Web if they are to truly represent added value to Web users. This includes issues of persistent storage, efficient reasoning, data mediation, scalability, distribution of data, fault tolerance and security.

In this paper [1] we present a coordination middleware for the Semantic Web and demonstrate its relevance to these vital issues for Semantic Web applications. We introduce a typical use case in which an intelligent traffic management system must support coordinated access to a knowledge base for a large number of agents. Through a requirements analysis and a consideration of the state of the art we note that current approaches can not adequately support such an use

[1] This work is partially supported by the EU Network of Excellence KnowledgeWeb (FP6-507482)

case and propose a new solution based on the Linda coordination model [8]. Our design and implementation approach for a 'Semantic Web Space' is described, and the operation of the 'Semantic Web Space' is illustrated through examples from the intelligent traffic management use case.

2 Overview of the scenario

The Semantic Web is being evaluated and tested in a number of application fields in which it is expected that semantic enhancements can lead to added value for users and implementers. These fields include Web Services [17, 21, 3, 7, 18, 19], Grid Computing [10, 4, 5] and Multi-Agent Systems [24, 12, 9, 3].

In this paper we introduce a sample use case in the field of Multi-Agent Systems. We chose a traffic management scenario as it is a typical application domain for multi-agent systems which requires particular communication capabilities with support for coordination between a large number of agents. A generic scenario in this field proposes a large number of agents, both mobile (vehicular) and static (traffic controllers such as traffic lights or message systems), sending and receiving data to and from a central data store. Typically such a system, built according to the principles of multi-agent systems, has a high level architecture similar to the one illustrated in Figure 1.

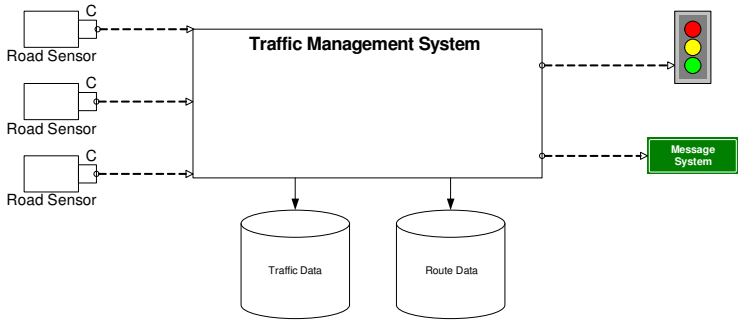

Fig. 1. Architecture of common traffic management systems

Multiple sensor agents collect and send traffic data to the system which is stored in a back-end database. In combination with route data, the system is able to interpret the data and send messages to traffic controlling agents such as traffic lights and message systems at certain locations in order to control the traffic flow (e.g. relieve congestion by holding back traffic or divert traffic around a bottleneck). In addition to this, mobile agents (i.e. vehicles) could access the system to request routing information. The system would not only calculate the possible routes from A to B on the basis of the back-end route data, but also take into account available traffic data to determine the best route at the time

of request based on quantitative criteria such as estimated journey length and duration. We expect both traffic and route data to be expressed in very low level terms, e.g. spatial position expressed using GPS coordinates and traffic conditions expressed using numeric measurements.

The Semantic Web and multi-agent systems have been identified as being complementary [12, 24]. In particular adding semantics to Web Services is likened to intelligent agent-based approaches [9, 3]. In this context, the agents are seen as self-contained applications which exhibit characteristics typical of applications on the (Semantic) Web: they are many, distributed, dynamic, heterogeneous and non-persistent. Hence we extend the traditional traffic management use case with semantics by expressing traffic and route data unambiguously in terms of a shared, common conceptualization of the domain, i.e. an ontology. This permits reasoning over the knowledge of the traffic management system to deduce new information which is of use to agents.

In the extended scenario we add ontology-based knowledge about vehicles, roadways and points of interest. Mobile agents are identified according to their location and vehicle type and can request routing information to specific points of interest. Hence the traditional traffic management system functionality is extended by the semantics and efficiently supported through Linda-like coordination. Traffic control can then take into account use cases such as restrictions on vehicle type (e.g. that a tractor can not travel on a motorway) and support queries based on reaching some specified type of service (e.g. such as a petrol station) in the most efficient means possible (e.g. inferring when and where traffic conditions are best).

3 Requirements for Semantic Web Technologies

As underlined in the previous section the usage of Semantic Web technologies in multi-agent systems provides significant advantages especially when these systems are enhanced with semantically represented domain information which can be used as shared vocabulary among agents or for inference purposes. However besides appropriate Web-compatible representation languages Semantic Web technologies need a powerful middleware for information management and agent communication, which is able to deal with typical characteristics of Web-based applications. We identified several requirements for an efficient Sematic Web middleware:

- a decentralized and distributed architecture, in order to allow agents to publish and retrieve information efficiently and effectively.
- scalability as a central issue because of the dimensions and the dynamics of the Web-based multi-agent scenario.
- a high-level of abstraction to cope with inherent heterogeneity problems.
- support for asynchronous interaction among agents and between agents and the middleware. Interaction should be uncoupled in space and time in order to allow agents to publish and retrieve traffic information in a flexible and efficient manner.

A second category of requirements relates directly to the representation and processing of scenario-relevant domain knowledge; there is a need for representation languages which are able to describe 'static' domain knowledge, like types of traffic agents, points of interests, traffic conditions. Appropriate representation languages should be provided to formalize 'dynamic' information e.g. recent traffic flow conditions, current events related to specific points of interest. Reasoning engines able to deal with the two types of knowledge are indispensable for the realization of intelligent multi-agent systems.

4 Semantic Web Technologies Today

This section analyzes the state of the art in Semantic Web research w.r.t. the use scenario requirements from Section 3. The Semantic Web [1] aims to provide automated information access based on machine-processable semantics of data. The first steps in this direction have been made through the realization of appropriate representation languages for Web knowledge sources like RDF(S) and OWL and the increasing dissemination of ontologies, that provide a common basis for annotation and support automatic inferencing for knowledge generation. A Description Logics-based language like OWL can be used to represent the so-called "static" domain knowledge required by the traffic management scenario (see Section 7). However formalizing "dynamic" knowledge like temporal information requires more expressive representation techniques which are not supported in a standardized manner by the Semantic Web Community. Rule languages for the Semantic Web have been proposed in several approaches [14, 15], but the interoperation between the (OWL-based) ontology layer and the consecutive rule layer is still an open issue.

The storage and processing of traffic information should be realized using a high-level, distributed middleware which permits agents to insert and retrieve information in an flexible and efficient manner. Such a middleware copes with the heterogeneity of specific storage systems for Semantic Web data [2, 11, 13] and offers a simple interface for the agents to get access to the information. Currently Semantic Web technologies do not address these aspects in a satisfactory manner.

Communication and interoperation are crucial characteristics of our scenario. Currently interaction in Semantic Web applications is based on the classical client-server model and message exchange requiring strong coupling in terms of reference and time. The communication needs to be addressed to the communicating parties and it is synchronous. As mentioned in Section 3 the traffic management scenario as well as a much broader area of multi-agent applications or Web Services [6] require different communication paradigms to realize the envisioned Semantic Web.

As a conclusion of this section we underline that Semantic Web technologies do not cover the requirements of real world Web-based multi-agent systems to a satisfactory extent. We propose a 'Semantic Web Space' with an underlying tuplespace paradigm as a possible solution for an open, distributed, scalable midlleware for the Semantic Web (see Section 6).

5 Linda and TupleSpaces

Before describing the key concepts of the Semantic Web Space we introduce the foundations of our approach, the coordination language Linda and the tuplespace paradigm, and discuss how they fulfill the requirements mentioned in Section 3.

The coordination language Linda [8] has its origins in parallel computing and was developed as a means to inject the capability of concurrent programming into sequential programming languages. It consists of a shared data space (*the tuplespace*) which contains data (*the tuples*) and coordination operations (*the coordination primitives*) that are applied in the shared data space.

The tuplespace is a shared data space which acts as an associative memory[2] for a group of agents. A tuple is an ordered list of typed fields and retrieval is governed by matching tuples against a template which is a tuple containing both literals and typed variables. A match occurs when the template and the tuple are of the same length and the field types and the value of constant fields are identical. For example, if a tuplespace contains the tuple

```
("Bobby Bear", GBP, 25.18)
```

then it will match a template such as

```
("Bobby Bear", GBP, ?amount)
```

with the value 25.18 being bound to the variable 'amount' [3].

The coordination primitives are a small yet elegant set of operations that permit agents to emit a tuple into the tuplespace (operation *out*) or associatively retrieve tuples from the tuplespace either removing those tuples from the space (operation *in*) or not (operation *rd*). Both retrieval operations are blocking, i.e. they return only when a matching tuple is found. In this way Linda combines synchronization and communication in an extremely simple model with a high level of abstraction.

The following features of Linda have been mentioned as attractive for programming open distributed applications [20]:

- It uncouples interacting processes both in space and in time. In other words, the producer of a tuple and the consumer of that tuple do not need to know one another's location nor exist concurrently.
- It permits associative addressing, which means that data is accessed in terms of what kind of data is requested, rather than which specific data is referenced.
- It supports asynchrony and concurrency as an intrinsic part of the tuplespace abstraction.
- It separates the coordination implementation from characteristics of the host platform or programming language.

[2] Associative retrieval implies that tuples are not addressed by ID or address, but by their content.

[3] But there will be no match with ("Polly Panda", GBP, ?amount), ("Bobby Bear", EUR, ?amount) or ("Bobby Bear", GBP, ?amount, "DiscountStock").

Several Linda implementations as well as extensions of the core concept have emerged in the last decades. We mention in particular XMLSpaces [23] which is our extension of the basic Linda model to support the manipulation of XML documents within tuple fields.

6 Semantic Web Space

The Semantic Web Space is a middleware platform intended to fulfil the requirements of reliability, scalability, self-organization, coordination w.r.t. the open distributed system of the Web. We plan to make an initial prototypical implementation as an extension of our tuplespace platform XMLSpaces.

Semantic Web-based systems make use of access to knowledge stores distributed on the Web to acquire and infer knowledge required for specific tasks. These knowledge stores must handle parallel access from multiple, heterogeneous systems, coordinating responses with other systems (e.g. that resolve ontological mismatches). Applying tuplespaces to the open global environment of the Web raises new requirements, some of which have already been mentioned in other work [6, 16]:

- The naming of spaces, semantics and structure in describing the information content of the tuples. Otherwise tuples can not be distinguished from one another in terms of their contents when they have the same number of fields and field order.
- The nesting of tuples. Web data models such as XML permit the nesting of elements within a single document. Likewise Web-based information should be able to explicitly show where one unit is contained within another.
- A reference mechanism. The Web uses URIs as a global mechanism to uniquely address resources.
- A separation mechanism. Distributed applications which have independent naming schemes may use the same names for their tuplespaces, semantics or structure. On the Web, vocabularies can be kept separate – even when using the same terms – using the namespaces mechanism.
- Richer typing. Tuple values are typed according to the core data types. However this is not precise enough in a large scale environment with dynamically changing information. Richer typing can support validation and correct interaction with tuplespaces.

In a Semantic Web Space we represent RDF statements as tuples and the RDF graph as a tuplespace. Each tuple in the tuplespace has three fields typed rdfs:Resource, rdf:Property and rdfs:Resource, modelling the RDF triple. All RDF resources are represented in tuple fields by URIs. In order to support rich typing, tuple fields are also typed using URIs identifying classes constrained by an (RDFS/OWL) ontology. In other words, each field value in the tuplespace is associated to a RDF type. We consider three RDF modelling primitives in particular in terms of their representation within a Semantic Web Space. (See also [22]).

- **Blank nodes** are nodes in the RDF graph which do not have any form of URI identification. However associations tied to the same blank nodes must be both maintained in the tuplespace as well as represented by clients writing tuples or to clients reading tuples. We propose an extended RDF type called BlankNode, which can be instantiated in a tuple and given an internal unique URI value for representing that blank node in the local tuplespace, playing the role of "blank node identifier".
- **Containers and collections** are special RDF objects which represent sets of resources. We propose that the members of rdfs:Container typed resources (rdf:Bag, rdf:Alt, rdf:Seq) are represented by a resizable array datatype, and that the members of a collection (rdf:List) are represented by a closed array datatype. In the tuplespace the container or collection can be referenced as a blank node or by an URI, and is related to its members through a statement with the rdfs:member property.
- **Reification** is the means by which a statement can be made about another statement in RDF. For this, RDF uses its own vocabulary (rdf:Statement, rdf:subject, rdf:predicate, rdf:object) to identify a statement with an URI so that it can be used as a subject or object of another statement. We propose the use of nested tuples to represent reification in the tuplespace. In this way a statement (as a tuple) can be the value of a field in another tuple. Nested tuples are considered as instances of the RDF type rdf:Statement with an internal or global URI as identifier.

In order to retrieve a tuple from the tuplespace new matching relations shall be implemented based on ontological reasoning on classes and properties. Different URIs may refer to the same concept, and different concepts may be identifiable as sub- or super-types of the given concept. Matching should be possible at different levels of precision and should have access to and make use of ontological knowledge to determine the relationships between classes and instances in the tuplespace. A mapping between a RDF/OWL query language and tuplespace templates must be defined (see Section 7)

7 System design

The requirements analysis of the traffic management scenario (see Section 2) revealed that Semantic Web technologies do not currently cover some significant aspects related to coordination and scalability. For the realization of real world Semantic Web-based systems, one needs powerful middleware technologies to cope with these typical characteristics of open distributed infrastructures as the Web. In this section we propose an extension of the Semantic Web enhanced traffic management scenario into tuplespace computing. The Linda model for coordination is suited to this scenario, as it provides the basic requirements of the system: a common data store, support for multiple agents and their interaction, coordination of that interaction and decoupling from time and space. A new architecture is proposed (Figure 2) in which the agents interact directly

with the data store using the simple Linda co-ordination primitives[4]. The functionality of the system is also abstracted into external agents who interact with the data store. This not only is a basis for modularizing the traffic management system and hence supporting reusability and updatability but also makes system knowledge directly available to any interested (and access-enabled) agents. Simple agent operations (reading some knowledge from the system) are then standardized (through Linda) and supported from the tuplespace platform without requiring any specific functionality to be executed from the traffic management system.

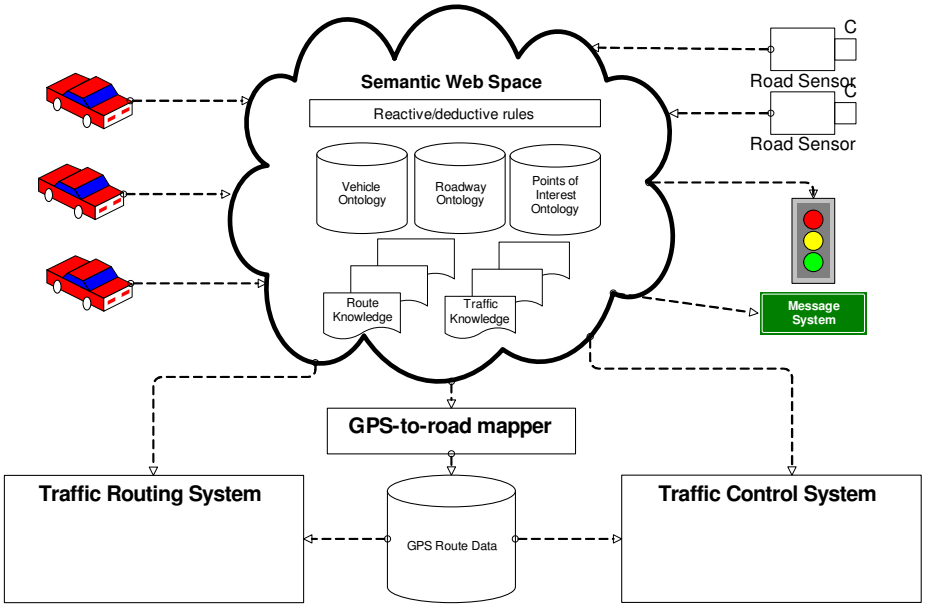

Fig. 2. Tuplespace-enhanced architecture of a traffic management system

To illustrate the use of ontologies and tuple-space functionality we consider three use cases: (1) A vehicle requests a route not to a specific location but to a point of interest (2) Routing takes into account the characteristics of the vehicle (3) Routing takes into account characteristics of the route being proposed.

7.1 Point of interest Use Case: Getting to the nearest Esso station

In this use case, low level routing data is not sufficient to meet the routing request. The system needs to be able to understand what the user is asking for

[4] Extensions to this primitive set will be considered later. At this stage we consider that 'in' and 'rd' operations retrieve all matching tuples without addressing further in this paper the 'multiple read' problem.

(i.e. what is an Esso petrol station) and have access to the information about it (i.e. where Esso petrol stations are located). Hence we must extend the system with an ontology for points of interest and a knowledge base which defines instances of the location types in the ontology and tie them to physical locations (co-ordinates). Once the system resolves the locations, it can calculate the alternative routes and suggest the best route on the basis of distance/expected duration. This knowledge could be expressed within the system in a structured way (e.g. records in a database) but without the logical model that would support reasoning over a request.

For example, there might not be an Esso petrol station anywhere nearby. The user is requesting Esso because he has a loyalty card, but this card is valid at Total petrol stations too (for the purposes of this scenario we suppose that they are brands of the same company). So with that knowledge the system could infer that the user won't mind being directed to a Total station if it is nearby. When none of these brands are close, the system should also be able to infer that the user will then accept being directed to some other station (he loses out on loyalty points but at least he can still buy petrol).

For example, a small ontology of petrol stations can be formalized in Description Logics as follows:
$PetrolStation \sqsubseteq \exists belongsTo.Company$
$Company \sqsubseteq \exists hasCustomerProgram.CustomerProgram$
$Company(Esso)$
$Company(Total)$
$CustomerProgram(PayBack)$
$hasCustomerProgram(Esso, PayBack)$
$hasCustomerProgram(Total, PayBack)$

An agent requesting this route sends a message with its location and a statement "find me the route to an Esso petrol station" like this[5]:

```
(1) Out(#agent876[tms:Agent],loc:isAt,
    "long=04657459345&lat=47856486475"[geo:GPS])
(2) Out(#agent876[tms:Agent],loc:routeRequest,
    (?X[poi:PetrolStation],poi:belongsTo,poi:Esso))
```

- Note that the variable ?X in (2) is constrained not only by its type but also as a subject in a nested template to the matches made to that template. In other words, ?X is matched to instances which are of type poi:PetrolStation AND which belong to the company Esso.
- As a result a set of tuples are inserted in which the object of the route request is a Esso petrol station. In other words the nested template sent by the agent acts as a representation of a set of matching instances.

[5] A QName represents a class or instance in an ontology, a term in speech marks is a literal, a value beginning with a hash is an internal ID, a value beginning with a question mark is a variable and a value in square brackets is the field type.

It would be the role of the system functionality in the Traffic Routing System to monitor for a tuple with a loc:routeRequest property (i.e. a routing request) (3) and when a match is returned to query in the tuple space for the GPS location of the requesting agent (4) and the desired point of interest (5) through these templates:

```
(3)Rd(?A[tms:Agent],loc:routeRequest,?P[poi:PointOfInterest])
(4)Rd(?A[tms:Agent],loc:isAt,?START[geo:GPS])
(5)Rd(?P[poi:PointOfInterest],loc:isAt,?END[geo:GPS])
```

Within the Traffic Routing System the possible routes between the GPS coordinate values tied to the variable ?START and the variable ?END can be calculated. The selected route (e.g. based on distance) would be inserted into the tuplespace (6) and read by the agent who is now monitoring for a tuple with the loc:RouteResponse property and the agent ID as subject. It is proposed that the route is expressed as a RDF sequence of GPS co-ordinates, i.e. the agent can retrieve the reference to the Sequence instance and can then read over time the co-ordinates to guide it to the point of interest.

```
(6) Out(#agent876[tms:Agent],loc:routeResponse,#route876[rdf:Seq])
```

Through the additional ontological information it is possible to reason over alternative possibilities for the route which still fulfil the user's request:

- It could be expressed that in the matching rule that a match on an Esso petrol station, Total petrol station or an IFP petrol stations should also match on instances belonging to the other two companies. This could be done by stating that any petrol station whose company has the Payback customer program is to be matched equally.
- It is proposed to enable in the matching procedure an optional support for supersumption, i.e. permit a match on a superset in the event of there being no matches on a subset of the class. In this case, given that the Esso petrol station is considered a subset of all petrol stations (the property of belonging to Esso being considered a class restriction)we could support that in the event of no suitable petrol station being available a route is proposed to some other petrol station.

By associating characteristics of the user to points of interest, we can support routing information tailored to the needs of the user e.g. that a tourist will prefer a slower route that takes in more tourist sights than the fastest route available.

7.2 Vehicle Use Case: Routing a slow-moving truck

Again in this use case the low level data is insufficient. The current routing calculation is based on a simple determination of possible routes from A to B, and selection based on internal calculation (e.g. the shortest distance). However road and vehicle metadata is a relevant input to the route deduction process, as

e.g. a slow-moving vehicle should not travel on a motorway or a high load on a road with a low bridge crossing over it.

In this case, two ontologies are required: one for vehicle types and characteristics, and another for road types and characteristics. These ontologies then also are able to define what is logically consistent or inconsistent. A system processing possible routes with this information can then reject anything which contradicts its ontological knowledge.

For example, we could model the following ontological knowledge for a vehicle and a road:
$Truck \sqsubseteq Vehicle$
$Vehicle \doteq \exists hasCharacteristic.VehicleProperty$
$SlowMoving \sqsubseteq VehicleProperty$
$HighLoad \sqsubseteq VehicleProperty$
(a) $Truck \sqsubseteq \exists hasCharacteristic.SlowMoving$
$Truck \sqsubseteq \exists hasCharacteristic.HighLoad$
$Motorway \sqsubseteq Roadway$
$Roadway \doteq \exists hasCharacteristic.RoadProperty$
$Vehicle \sqsubseteq \exists travels.Roadway$
$LowBridge \sqsubseteq RoadProperty$
$HighSpeed \sqsubseteq RoadProperty$
(b) $Motorway \sqsubseteq \exists hasCharacteristic.HighSpeed$
(c) $\exists hasCharacteristic.SlowMoving \sqcap \exists hasCharacteristic.HighSpeed \sqsubseteq \bot$
$\exists hasCharacteristic.HighLoad \sqcap \exists hasCharacteristic.LowBrodge \sqsubseteq \bot$

When generating a route for an agent the Traffic Routing System can check if the roadways travelled along in the route is consistent with the agent in terms of their characteristics. Note the requirement here for a GPS-to-road mapping component which is able to provide the necessary translation between GPS co-ordinates (which can be interpreted by the traffic management system) and roadway instances (which are understood by the Semantic Web Space). For a given route then the agent provides its characteristics (1) and an individual roadway in the route (2).

```
(1)Out(#agent876[tms:Agent],owl:sameAs,#truck876[veh:Truck])
(2)Out(#truck876[veh:Truck],loc:travelTo,
       "long=04657459367&lat=47856486511"[geo:GPS])
```

Note that the Traffic Routing System is only able to send tuples with GPS co-ordinates (as that is all that it understands). The GPS-to-road mapping is triggered by the GOS-to-road mapping agent monitoring for tuples stating that vehicles travel in some GPS co-ordinates (3), removing this tuple (2) and inserting into the tuple space a new tuple (4) with a Roadway instance value.

```
(3)In(?WHO[veh:Vehicle],loc:travelTo,?GEO[geo:GPS])
(4)Out(#truck876[veh:Truck],loc:travelTo,#road378645[road:Motorway])
```

Note that this statement is now inconsistent, referring to the ontological statements above: #agent876 is a particular instance of a truck, thus it can be inferred that it is slow moving (a), and is travelling onto a motorway. Since a slow moving vehicle can not travel on a motorway (b,c), #agent876 can not travel on a motorway.

The Traffic Routing System is informed that the statement it provided to the tuplespace (2) is logically false. A possible means to achieve this is that the Linda coordination primitives are extended to include the idea of checking the truth of a tuple. Then a "rdiftrue" or "iniftrue" can be used to retrieve a tuple if and only if the statement made in the tuple can be held to be logically true i.e. ontologically correct. Here the Traffic Routing System has inserted into the tuple space a set of possible roadways and now ins-if-true those tuples with a Roadway instance. A logically false tuple like the one above will not be read by the system, hence that possible route will not be considered any further.

```
InIfTrue(#truck876[veh:Truck],loc:travelTo,?RD[road:Roadway])
```

Vehicle and road metadata can be extended according to what is possible to determine from sensors or acquire from users. For example, rather than static characteristics of a vehicle such as its type, we could include information about the remaining petrol in the tank or the existence of children among the passengers, and hence in the route deduction include a petrol station along the route and prefer one with facilities for children such as a play area.

7.3 Route Use Case: Checking the traffic on the whole route

Finally in this use case we integrate the traffic conditions data being fed into the tuple space from the road sensors. In a traditional traffic management system, this data is processed by the application at a low level and results communicated to traffic control agents such as message boards or traffic lights. For example, where traffic is registered as being at a standstill diversions are placed into effect or where congestion is identified traffic may be held back.

The important aspect of traffic control data is that it is dynamically changing in real time. In a routing situation where vehicles are to be routed to their desired destination while taking into account real time changes in traffic conditions, the coordination functionality of the tuplespace is necessary. We consider a scenario where vehicle agents are constantly updating their position, are being routed to a particular destination, and are seeking to always take the fastest-moving route.

In this case, the Traffic Routing System carries out a further selection phase after checking all possible roadways for logically consistency (e.g. that no tractor is sent onto the motorway). From the set of logically consistent roadways, traffic condition data is retrieved and the system selects the roadway with the fastest-moving traffic. This is done by retrieving the instance of a traffic sensor at the given roadway and matching on the tuple which states the current traffic speed reported from that sensor e.g.:

```
Rd((?R[tr:trafficSensor],loc:isAt,#road0857[road:Street]),
tr:hasTrafficSpeed,?I[xsd:integer])
```

The agent could then perform the remaining functionality of checking which street has the fastest moving traffic. Hence the traffic management system can perform intelligent routing in that:

- 1. The mobile agent updates the tuple space with its current location
- 2. The traffic router updates the tuples containing the potential routes to the agents desired destination
- 3. Mobile agent and route characteristics are used to identify logical inconsistencies in routes, which are ignored by agents by using a "truth" test
- 4. From the remaining route possibilities, the agent selects the quickest route using traffic conditions also being expressed in the tuple space.

As all the knowledge for the routing and traffic management is being stored in the tuple space, agents can act upon the 'overall' view of the data even when some data is spatially or temporally disjoint (i.e. the originating agent is no longer available or the inserted tuple was placed into the space at an earlier timepoint). For example, in the Linda model the *rd* operation is blocking, meaning the template only returns when a match is found. This can be used by an agent to wait for a notification when the entire route is good to take:

```
Rd((?R[tr:trafficSensor],loc:isAt,(#route876[rdf:Seq],
rdfs:member,?ST[road:Roadway])),tr:hasTrafficCondition,
"GOOD"[tr:trafficCondition])
```

Note that the agent can access all roadways in a route in that a route is modelled as a RDF sequence and hence all its members can be accessed through the rdfs:member property. Additionally, the traffic condition with the controlled vocabulary value "GOOD" exists to simplify reasoning over the traffic statistics being generated from the traffic sensors. Statements with these conditions could be inserted from the Traffic Control System inferred from the traffic statistics that it reads from the tuple space. A possible extension would be to consider points of interest in the vicinity of the route and events associated with them that lead to changes in traffic conditions. Then the inference by the Traffic Control System could be extended to include predicted conditions based on where and when the agent will be travelling. For example, the major routes to and from a football stadium are likely to be busier at times shortly before and after a football game. Events could be integrated from other sources on the Web like a football league schedule. A similar case would be to permit the insertion of traffic announcements from other sources such as accidents (from the emergency services) or building works (from the public roads department), and to be able to reason on the consequences for traffic on nearby roadways (e.g. if a given road is being closed off, traffic on a parallel route will shortly increase) and include this reasoning in the routing calculations. Importantly this use case raises the need for extensions with spatial and temporal ontologies and rules as well as probabilistic logic.

8 Conclusions and Future Work

In this paper we presented the usage of the tuplespace paradigm as Semantic Web middleware for a traffic management system. Tuplespaces are a good alternative to common information management and interaction models on the Web, since they allow agents to publish and retrieve information in an uncoupled manner in terms of space and time. By extending tuplespaces to represent Semantic Web knowledge we allow Semantic Web applications to store and exchange information in a decentralized and distributed manner, while taking advantage of the powerful coordination mechanism of Linda. However the realization of Semantic Web enhanced tuplespaces means not only enabling RDFS(S) and OWL data to be represented in terms of tuples, but also a revision of the classic Linda model w.r.t. the meaning of its primitives. A redefinition of Linda primitives in the context of the Semantic Web Space is subject of future work.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 5 2001.
2. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *The Semantic Web - ISWC2002*, 2002.
3. Paul Buhler and José M. Vidal. Semantic Web Services as Agent Behaviors. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, editors, *Agentcities: Challenges in Open Agent Environments*, pages 25–31. Springer-Verlag, 2003.
4. Mario Cannataro and Domenico Talia. The Knowledge Grid. *CACM*, 46(1):89–93, 2003.
5. D. De Roure and J.A. Hendler. E-Science: the Grid and the Semantic Web. *IEEE Intelligent Systems*, 19(1):65–71, 2004.
6. D. Fensel. Triple-based computing. http://www.wsmo.org/2004/tp-computing/, June 2004.
7. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.
8. David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.
9. Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. Agent-based semantic web services. In *Proceedings of the twelfth international conference on World Wide Web*, pages 710–717. ACM Press, 2003.
10. Carole Goble and David De Roure. The Semantic Grid: Myth Busting and Bridge Building. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, Valencia, Spain, 2004.
11. S. Harris and N. Gibbins. 3store:Efficient Bulk RDF Storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, 2003.
12. J. Hendler. Agents and the Semantic Web. *"IEEE Intelligent Systems*, 16(2), 2001.
13. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, 2004.

14. Ian Horrocks and Peter F. Patel-Schneider. A Proposal for an OWL Rules Language. In *the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
15. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *Available at http://www.w3.org/Submission/SWRL/*, 2004.
16. B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. *Journal of Systems and Software*, 69(3):243–266, 2004.
17. R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic Web Services: description requirements and current technologies. In *International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003)*, Pittsburgh, PA, 2003.
18. Enrico Motta, John Domingue, Liliana Cabral, and Mauro Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. http://www.cs.unibo.it/ gaspari/www/iswc03.pdf, 2003.
19. OWL Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, November 2003.
20. Davide Rossi, Giacomo Cabri, and Enrico Denti. Tuple-based Technologies for Coordination. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 4, pages 83–109. Springer Verlag, 2001. ISBN 3540416137.
21. Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth, and John Miller. Adding Semantics to Web Services Standards. In *Proceedings of the International Conference on Web Services (ICWS'03), 2003*, June 2003.
22. R. Tolksdorf, L. Nixon, F. Liebsch, N. Duc Minh, and E. Paslaru Bontas. Semantic Web Spaces (Technical Report TR-B-04-11). Technical report, Free University of Berlin, 2004.
23. Robert Tolksdorf and Dirk Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems (CoopIS 2001)*, number LNCS 2172, pages 356–370. Springer Verlag, 2001.
24. Yououng Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan. Using Semantic web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment. *Proceeding of the 5th International Conference on Electronic Commerce*, September 2003.