A Conceptual Model for Semantic Web Spaces

Elena Paslaru Bontas, Lyndon J. B. Nixon, Robert Tolksdorf
paslaru, nixon, tolk@inf.fu-berlin.de
Freie Universität Berlin
Institut für Informatik
AG Netzbasierte Informationssysteme
Takustr. 9, D-14195 Berlin Germany

1 Introduction

In a previous technical report [TNL+04] we have introduced an extension of XML-based tuplespaces for the Semantic Web. By applying a tuplespace approach to the concurrent interaction of multiple clients with distributed knowledge repositories, we foresee the benefits of a simple yet powerful co-ordination model in which parallel and distributed processes can be uncoupled in space and time. The proposed system has been called Semantic Web Spaces and is envisaged as acting as a middleware platform for real world Semantic Web applications [TPBN05, TNPB+05b], in which it can handle in place of the client the administration of distributed data, co-ordination of multiple processes and mediation between ontological representations. This technical report outlines a conceptual model for Semantic Web Spaces. It provides answers to the questions that arise when trying to combine the Linda/tuplespace paradigm with the Semantic Web and Semantic Web compatible knowledge representations. Considering the sorts of extensions that can be made to Linda-based systems, we identify those extensions which are required for an integration with the Semantic Web. As a result, we can give a clearer overview of how Semantic Web Spaces would be implemented and function. As a complementary activity, we also provide in a separate technical report a concrete use case for applying Semantic Web Spaces to realize ontology repositories [TNPB05a].

2 Overview of the conceptual model

Since its original conception as an extension to sequential programming languages to enable parallel programming, Linda [Gel85] has undergone a multitude of different extensions in order to apply it in a wide range of scenarios which benefit from its simple yet powerful co-ordination model (see Chapter 2 of [TNL+04] for a more detailed description of Linda and Linda-based applications). In particular, recent work has drawn up requirements for its application in open distributed systems [JF04].

Following the categorization drawn up in [RCD01], our proposed extensions for tuplespaces can be divided in 4 categories:

- New types of tuples: the representation of Semantic Web knowledge within tuplespaces requires new types of tuples which are tailored to common Semantic Web languages such as RDF(S) [HM04], OWL [PSHH04] or SWRL [HPSB$^+$04].
- New co-ordination primitives: the transit from classical data-centered tuplespaces to the new semantics-aware Semantic Web Spaces requires a revision of the meaning of the core Linda primitives and the definition of further operations (see Section 4).
- New matchings: the standard Linda matching approach has to be extended in order to efficiently manage the newly defined tuple types. Semantic matching techniques taking into account ontological knowledge can be used to enrich the retrieval capabilities of the tuplespace.
- Tuplespace ontology: the tuplespace ontology is used as a formal description of the tuplespace, its components and properties. Using ontologies in this context allows a more flexible and efficient management of the tuplespace content and of the interaction between tuplespace and information providers and consumers (extendability, automatic inferencing etc.).

Figure 1 below shows an abstract architecture of Semantic Web Spaces. As all Linda-based systems, the central components are the Linda co-ordination model and the tuplespace as a shared data space for tuples.[1] In Semantic Web Spaces we extend the core architectures with a reasoning component for interpreting ontologies according to their formal semantics (and drawing inferences, checking satisfiability etc) as this is out of the scope of the Linda paradigm. Accordingly, the tuplespace is extended to support building a semantic view upon the tuples (i.e. construction of a RDF graph model from RDF data stored in the tuplespace) and association of RDF statements with the ontologies they reference.

Additionally, we extend the component handling the co-ordination of processes with modules to fulfil different administrative services that we consider as requisites in a Semantic Web middleware [TNL$^+$04, TNPB$^+$05b]. We exemplify this issue by considering security and trust components as extensions of the classical architecture. A set of metadata (e.g. RDF statements) describes the tuplespace itself, according to an ontology we define for describing a tuplespace and the tuples that it contains. This ontology provides concepts for expressing security and trust policies, and hence allows for an ontology-based approach to organizing and initializing these extension modules. Further on, the ontology explicitly describes the structure of the space (e.g. whether sub-spaces are allowed) and the supported matching templates.

Finally, as the system is foreseen as a middleware platform, it should be independent of the underlying implementations of the different computer systems that the system must interact with. This necessitates interfaces to isolate the system kernel from the heterogeneity of both the clients which communicate with the system and the backend storage solutions

[1]The “kernel” of such a system could be distributed, the diagram does not purport to say anything about the actual location of components.

which realize the physical storage of the information represented in the logical memory of the tuplespace.

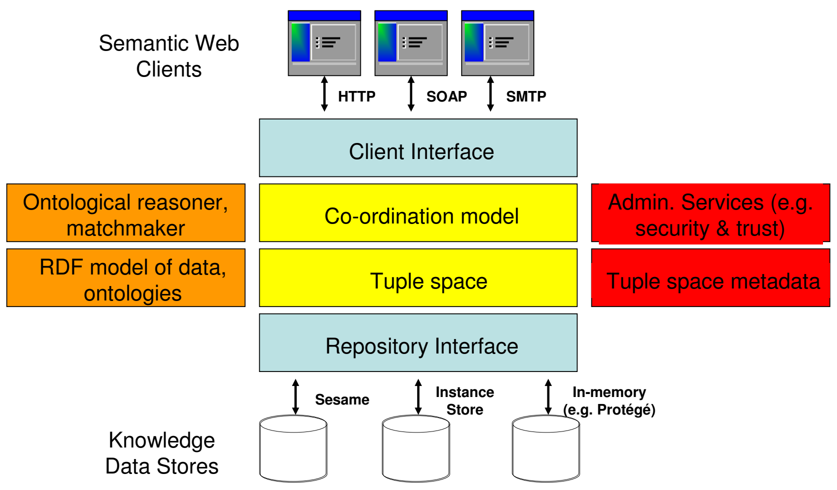

Figure 1: High-level architecture of Semantic Web Spaces

We now turn to a description of the tuplespace extensions in the aforementioned categories: new types of tuples, new primitives, new matchings and tuplespace ontology.

3 New types of tuples

Following the Linda paradigm, Semantic Web Spaces should be able to represent *Semantic Web information* through *tuples*. The expressivity of the information representation should be aligned to the expressivity of common Semantic Web languages, while respecting their semantics, so that tuples could be mapped to and from external Semantic Web resources. As Semantic Web languages, we focus on RDF(S) [HM04] and OWL [PSHH04]. RDF triples can be represented in a three field tuple of form <subject, predicate, object>. Each tuple field contains, following the RDF abstract model, an URI (or, in the case of the object, a literal). These URIs identify RDF resources. Each field is also typed accordingly, URIs by RDF/OWL Classes and literals by datatypes. Finally, we choose to add an ID field to the tuple. This allows each tuple to be globally identified. In this way tuples sharing the same content i.e. the same subject, predicate and object can be addressed separately. The allocation of the tuple IDs is coordinated by the tuplespace, while clients are permitted to assign IDs within particular namespaces. [2]

Hence every RDF statement can be represented in a tuple and the type information carried in each field. For example, a RDF tuple (using QNames instead of full URIs) stating that a certain book has a certain ISBN number could look like this:

```
<my:novel [books:Book], books:has-isbn,
 isbn:508-1762-443 [books:Isbn], my:bookisbn>
```

We also define a tuple field type “Bottom” which is represented in the syntax as underscore “_”. ’Bottom’ represents the empty set of things, the underclass of objects to which no object belongs. It is reasonable to assume that such a thing exists following set theory (the axiom of empty set). While not in RDF, the empty set is defined at the level of OWL (owl:Nothing).

For templates, we define two further concepts. The first is that template (but not tuple) fields may contain variables, represented in the syntax with a question mark, followed by a type. Unlike traditional Linda, variables are not bound when tuples match to a template hence they do not need to be named. The second is that templates may contain wildcards, represented by an asterix. Wildcards represent the set of all things in the space, the class of objects to which all objects belong. Again, a similar concept is provided not in RDF but in OWL (owl:Thing).

There are however three special cases in the RDF model which we must also consider in terms of deciding upon a suitable tuple-based representation: blank nodes, collections/containers and reification. We consider each in turn.

3.1 Blank nodes

Blank nodes are anonymous resources in a RDF graph, which have no externally addressable label. However, blank nodes are assigned internal identifiers in order to allow other local statements to reference and distinguish them. Due to the local character of blank node identifiers, two RDF graphs which differ only at the blank node level are still considered equivalent. It is necessary to provide for the same semantics in Semantic Web Spaces.

We define a field type ts:BlankNode which identifies the use of a blank node in a tuple. The system itself will administer internal IDs for these blank nodes, ensuring that they remain unique within the tuplespace or within the scope of the tuple. For a client placing a tuple with a blank node into the tuplespace, the identifier it allocates to the blank node field is irrelevant, a local identifier is allocated to the field when the tuple is added

[2] In fact, since the tuplespace is described by means of an ontology, every tuple becomes an instance of the ontology class ts:Tuple and is therefore automatically identified by the URI assigned to this instance.

to the tuplespace. Again, since every blank node is defined as an instance of the class ts:BlankNode provided by the tuplespace ontology, the allocation of the corresponding ID is realized in a similar manner as for every tuple contained in the tuplespace. However, the IDs assigned to blank nodes are handled in a different way by the tuplespace itself in order to ensure their locality. To add multiple tuples which refer to the same blank node, it is necessary to claim all the tuples together in the form of a subspace, see Section 4.1.

Hence a RDF tuple with a blank node could look like this:

```
<my:person [foaf:Person], vcard:address, _:80bg42c
[ts:BlankNode], my:personsaddress>
```

3.2 Collections and containers

A collection is a closed set of objects. Containers are incomplete sets built according to the open world assumption. Collections are specified in RDF by List, containers by Bag (unordered), Seq (ordered) or Alt (only one member is considered 'true' at any time). The RDF Semantics does not impose any special conditions on the defined RDF vocabulary, which permits some flexibility in the tuple representation but also means care must be taken in what is permissable.

In traditional Linda implementations, the constructs of the host programming language are used to define data structures in the tuples. In this case, we consider the use of array constructs to model collections and containers in tuples. This has the advantage of permitting them to be contained in a single tuple, independent of the collection/container size. Constructs would be needed to differentiate between static and dynamic as well as ordered and unordered arrays.

A List is then modeled in a static, ordered array. This replaces a set of rdf:first/rdf:last statements. Rather, the client can retrieve the List as an array object and handle it locally using host programming language constructs.

While not explicitly modeled as a tuple, the implicit property of membership between a List and any resource could also be queried using the template

```
<x [rdf:List], rdfs:member, ? [rdf:Resource], *>
```

meaning return any resource which is a member of List x. This template would only function in the information view of the space, and not in the data space. Given that List x was asserted as a tuple field and that the same statement has not been explicitly asserted in the data space, rdfs:member statements are only implicit in the space and can be made explicit only if considering the semantics of the RDF language. The data view of each tuplespace does not have any knowledge about the semantics of the data content, while administrating tuples built according to a particular syntax (e.g. RDF statements in triple form). Given the RDF syntax for Lists, we constrain Lists in the space to rdfs:member properties (i.e. rdf:first, rdf:rest and rdf:nil are not understood). This is because the client can iterate through a list locally, rather than have to do this through a cycle of templates querying the space with rdf:first and rdf:rest, and because it avoids the semantic difficulties which arise if a client could assert new values for rdf:first and rdf:rest in an existing List.

Containers are modeled in dynamic arrays, unordered in the case of Bag and ordered in the case of Seq and Alt (In the latter case, it makes it possible to choose a resource on the basis of its position in the container). Again, rather than a set of membership statements the client includes the array in the tuple field typed by the appropriate container type. Likewise, the container is retrieved from a tuple as an array and can be handled locally. The implicit property of membership between a container and any resource could also be queried using the template (where x represents an integer between 1 and x inclusive, and x is the size of the container):

```
<x [rdfs:Container], rdf:_x, ? [rdf:Resource], *>
```

As tuplespaces do not permit tuple updates, a client which wishes to alter a collection or container could destructively read the tuple with the array, locally alter the array and then assert a new tuple. This helps the system maintain uniqueness of membership properties i.e. that only one resource is the value of

```
rdf:_1, rdf:_2
```

and so on, and that for a container with x members their membership is asserted by the properties

```
rdf:_1...rdf:_x
```

We could permit the above templates, as destructive reads, to effectively act to remove a member from the array. The system would then re-organize membership properties accordingly (if a member is removed, the membership property of the next members is moved down one). Likewise, statements of membership properties could be asserted in the information view of the space to act as member insertions. Again, the system would re-organize membership properties accordingly (if a membership property is asserted a second time, the value of it and all following members is shifted up one). In the RDF semantics, it is permitted that any given membership property has more than one value, or that a membership property has no value. We change this in order to allow clients more certainty of acquiring the information they need, which in the RDF Semantics can not be guaranteed.

As collections and containers are asserted as arrays in tuple fields rather than URIs they are modeled in the RDF graph as blank nodes. In order to support global identification, which is also necessary to construct templates which query resource membership through the space rather than at the client level, it is possible to query at the tuple space ontology level for the ID of the relevant field in the tuple.

3.3 Reification

To reify a statement means to be able to refer to it in another statement. In RDF, this is achieved by creating an instance of rdf:Statement and giving values to its properties rdf:subject, rdf:property and rdf:object. The URI of the rdf:Statement instance is then used as subject or object in another statement.

The modeling of reification in the tuple space depends a great deal on our understanding of its semantics, and in RDF it is widely recognized that this is underspecified. We consider the reified triple (the rdf:Statement) not to be equivalent to any triple asserted in the space by a RDF tuple, so that the (fourth field) tuple ID can not be considered to be a means of reification. Rather, a reified triple represents a statement that is being talked about, and not any statement that has been made. Likewise, we do not assume that any reified triple of the form $<$s,p,o$>$ is equivalent to any actual triple $<$s,p,o$>$ even when s,p and o are URI equivalent.

In Semantic Web Spaces, we follow standard RDF syntax for expressing a reified triple. We refer to a rdf:Statement instance in a tuple, and give this instance values for the properties rdf:subject, rdf:property and rdf:object.

It is permissable to declare a Statement that is not fully instantiated, e.g. no rdf:object is expressed.

As a shorthand, a client could form a RDFtuple and add it to a tuple field typed as rdf:Statement. This tuple nesting avoids three additional tuples having to be asserted client side (they are of course asserted implicitly in the information view). As a RDFtuple it also contains an identifying URI within the fourth field. To model this in the tuplespace ontology rdf:Statement would be a subclass of ts:Tuple.

For example, to state my belief that one of my friends is currently in Australia, the tuple could look like this:

```
<my:person [foaf:Person], trust:believes, <my:friend3
[foaf:Person], loc:is-located-in, country:Australia
[country:Country], my:statementId> [rdf:Statement], my:claim>
```

4 New co-ordination primitives

The core Linda model consists of three primitives: out (tuple insert), rd (tuple retrieval, non-destructive) and in (tuple retrieval, destructive). Another original primitive eval has become less used in Linda implementations due to issues about its operational semantics. Other implementations have added their own primitives, for example as rd/in are blocking, non-blocking variants (rdp/inp) have been proposed. Another issue, the multiple read problem – Linda does not guarantee unique answers to the same template so that rd(template) could return the same single tuple repeatedly – is handled by the proposed collect and copy-collect primitives.

As aforementioned, in Semantic Web Spaces we make a fundamental distinction between a data view and an information view upon stored tuples. In the data view all tuples, despite their RDF-based typing, are seen as plain data, without semantics. Under this perspective the tuplespace works as a traditional Linda system. RDF tuples are not assigned any special meaning. In the information view we see the set of RDF tuples in the tuplespace as a RDF graph. This means that a RDF tuple must be consistent with the RDF semantics and satisfiable according to its RDF schema.

Hence, we can make two distinctions which both need a new set of Linda primitives, as each distinction determines a new semantic for the traditional Linda operations of out, rd and in. The first distinction is between non-RDF tuples and RDF tuples. A simple data tuple could also contain three fields with URIs as such tuples are unconstrained in terms of field number of field content. RDF tuples however have a special structure and components, which have to be handled according to the usage of RDF in the Semantic Web. So besides the traditional data tuple operations

```
out: tuple -> boolean.
rd: template -> tuple.
in: template -> tuple.
```

we redefine the primitives for RDF tuples, adding the constraint that the tuple or template that is used must conform to a RDF tuple or template (i.e. 3 or 4 fields, containing RDF resources or – as object – literals)

```
outr: (s,p,o) -> boolean.
rdr: (s,p,o,id) -> tuple.
inr: (s,p,o,id) -> tuple.
```

As mentioned above, the IDs are assigned internally by the system and are retrieved using special matching templates.

These operations are still data view, i.e. a RDF tuple conforming to RDF syntax is accepted, no check is made against ontological (e.g. RDF Schema) information. They serve to allow well formed RDF statements to be placed in the tuple space, and be retrieved destructively or non-destructively. As data view operations, templates act, in the sense of Linda, only on the syntactic level of the tuple, i.e. they can match on the basis of URI syntax matching or literal datatype matching. However, no ontological information is considered. This provides a less resource-intensive means to acquire RDF tuples, e.g. retrieval by ID, or by (exact) URI match, while a matching taking into account semantic information usually implies a considerable amount of computational resources required for performing particular inferencing operations.

However, for the information view, we want that the RDF statements are also satisfiable—according to a constraining schema—and retrievable based on ontological information. Information view assertion acts as a test of the consistency of the knowledge asserted in Semantic Web Spaces. RDF statements refer to a RDFS or OWL ontology, which describes the vocabulary and the meaning of these ontological constructs. For this reason RDF tuples which do not conform to the corresponding schema will be automatically rejected by the tuplespace. Note that we assume RDFS statements about the classes instantiated within the RDF tuple are available. If not, we reject the RDF tuple at the information view level as it can not be checked for consistency. The tuple could still be asserted as the data view level, and maybe asserted again later when RDF Schema information is available.

4.1 Claim

To assert tuples in the information view, we propose the primitive

```
claim: (s,p,o,id) -> boolean.
claim: (Subspace) -> boolean.
```

A claim can contain either a single RDFTuple (i.e. an instance of the class ts:RDFTuple, see below) or it can contain a Subspace, which is defined at the client side and contains one or more tuples. As well as allowing multiple tuples to be claimed in one operation, it provides the means to make claims which contain blank nodes. Within a Subspace, a blank node with the same identifier will be considered as being the same blank node when tuples are added into the space.

A claim carries a truth value, i.e. it is making a statement about something that it purports to be true. An accepted claim exists as a RDF tuple or set of RDF tuples equally in the data view, however its 'truthfulness' is only a property of the tuple at the information view level. If the claim can not be substantiated, then the tuple or subspace is rejected and false is returned to the client. Note that in the case of a subspace, the entire subspace must be satisfiable, or it will be rejected.

4.2 Endorse and excerpt

To read tuples from the information view, we propose the primitive

```
endorse: (s,p,o,id) -> Subspace.
```

A tuple matching the given template is considered 'endorsed' by the information view, i.e. that it has been found to be consistent with current ontological information. The match is returned as a subspace (see the tuplespace ontology: it is a tuplespace instance which is the object of the property 'hasSubSpace'). This subspace may contain a single matching tuple, however in the case of blank nodes the linking tuples are included in the response. This means for example that if the matching tuple has a blank node as its subject, then the set of tuples with the blank node as their object will be also included in the response.

We also choose to support a solution for multiple read operations (i.e. get all matching tuples for a template). We propose the primitive

```
excerpt: (s,p,o) -> Context.
```

This primitive is a version of the copy-collect primitive [RW98] that acts on the information view. It works within contexts (which are introduced next): a context is created by the system into which all matching tuples are copied [TPBN05]. A reference to this context is passed to the client who is given alone the right to access the context. The client can then make destructive reads in that context i.e. retract(*,*,*,*) to remove all of the tuples. When the context is empty the system destroys the context.

Note that endorse and excerpt can match tuples which are not in the data view but exist ’implicitly’ in the information view (i.e. as inferrable tuples).

4.3 Retract

Finally, tuples may be removed from the information view. Yet this operation is questionable, as it is not the same as expressing negation (which is not supported in Semantic Web Spaces). Rather, we are removing a statement from the set without denying its truthfulness. As a result, we propose the primitive

```
retract: (s,p,o,id) -> Subspace.
```

which, if a matching tuple is found, replaces its subject, predicate and object in the information view with ’Bottom’ rather than removing it completely. The tuple remains in the data view. Hence its reference continues to exist but the assertion that the reference makes is lost. As a result, all inferrable tuples from that retracted tuple must also be lost. Note that this operates the same as endorse, in that a subspace is returned and blank nodes will cause additional tuples to be retracted.

4.4 General issues

The information view primitives are, as with all Linda operations, blocking in order to support the co-ordination model of Linda. However, we do not consider this binding in an implementation of Semantic Web Spaces. There may be cases in which it is preferable to have non-blocking uses (where if a match is not found a null object is returned) or blocking with timeout.

We must also answer the issue of synchronization between data and information views. Fundamentally, the information view is a RDF schema-consistent view of a RDF graph built from RDF tuples in the data view. A claim made in the information view will also be added as a RDF tuple in the data view. However, a retraction in the information view does not affect the data view. Conversely, the assertion of a RDF tuple in the data view does not affect the information view—it is not considered as a knowledge claim.

The destructive reading of a RDF tuple in the data view does however alter the information view. The associated claim is retracted as a result, with the related consequences. Hence, in any application of Semantic Web Spaces, care must be taken in terms of which clients could destructively read which tuples.

There are two additional modeling issues that should be handled by Semantic Web Spaces. One is changes in the data and information view affect not only the tuple itself but also all tuples that are connected to it (e.g. exist in the sub-tree of the RDF graph for which that tuple is the root) or inferrable from it. Hence a single destructive read can have much larger consequences for the RDF graph in the Semantic Web Space. Similarly, changes in the ontologies being used to determine the satisfiability of RDF graphs may also cause tuples to be retracted. We do not consider these issues further in Semantic Web Spaces, but acknowledge that clients should be aware of the destructiveness of single operations and that in cases restrictions may be advisable on client operations to avoid such destructive operations.

We also choose to use the concept of contexts. Both clients and tuples are associated to a set of contexts, and an agent can only see the tuples which exist in a same context. The concept is based on scopes [MW00], which allow for a tuplespace to be split into (overlapping) partitions. This provides a means to control client access to statements in the space and to split client operations into subsets of relevant statements. In order to support contexts in the Semantic Web Space we require at least two operations: a means for a client to construct a context, and a means to acquire a context from a matched tuple. Other context operations are handled by the system e.g. if a client upon joining a space is allocated a certain context set.

At this stage in the conceptual model we do not define contexts further, but leave this as further work to be done.

The previous Linda operations must also be extended to support operations within a given context. The syntax used is operation(parameter)@context or contextset where context is a single context ID or contextset is an array datatype containing a list of context IDs, including nested arrays for context joins.

An operation asserting a tuple in the space which carries a context associates that context to the inserted tuple. An operation retrieving a tuple is executed in the system only over the tuples which are in the associated context.

5 New matchings

5.1 Syntactic and semantic matching

Traditional Linda matching is based on checking the equality of number of fields, equality of field constants and binding of field variables. Whether a matching occurs is therefore dependant upon matching rules for different types. Determining equality between two constants is type dependant as is determining if a constant in a tuple can be bound to the variable in the template. Initially simple datatypes were used as field types, and matchings were based on the equality checks of the programming language of the underlying Linda implementation. This is not necessarily as simple as stating that the types must be equivalent and that constant values are the same, e.g. the integer 5 and the real value 5.0 are equivalent, even though the type and constant value are actually different. For some types, different matching algorithms are possible, e.g. whether or not the lower/upper case of characters are ignored in string matching.

RDF can be handled both as a special datatype with a particular syntax (following the abstract syntax model) and as a knowledge representation form which encapsulates meaning about something (following the RDF model-theoretic semantics). Semantic Web Spaces support both analyzes of RDF data, through matching either in the data view or the information view.

Hence in Semantic Web Spaces we support three different levels of matching on the RDF tuples, which corresponds to the three sets of co-ordination primitives that are supported. The traditional Linda primitives (in, rd) match on RDF tuple contents disregarding RDF syntax entirely, i.e. RDF URIs are matched as if they were XML Schema anyURIs (and literals, of course, according to their own type). That is to say, URI equality is based on pure string matching and RDF URIs will type-match with any other string or anyURI typed URIs. While this will generally not be sufficient, it provides a least-computationally intensive approach to retrieving a RDF tuple. Special RDF syntactic constructs, in this matching, are disregarded: blank nodes, reifications or containers/collections are simply not understood at this level and can only match on a wildcard.

The other levels of matching, i.e. adding RDF specific syntactic matching (inr, rdr) and adding RDF Schema based semantic matching (endorse, excerpt, retract), are increasingly computationally demanding but of course provide the added power of the RDF data model and semantics to the clients of Semantic Web Spaces.

5.2 Matching RDF abstract syntax

The RDF syntactic matching considers only tuples identified in the space as RDF tuples. Every RDF tuple is constrained to the form <s,p,o,id>, whose 4 fields represent a RDF statement (subject, predicate, object) plus a tuple specific identifier. The RDF statement contains URI values typed as RDF resources (the object can also be a literal, i.e. a XML Schema datatype). The ID is also an URI which is a RDF ID. RDF resources are matched on the basis of URI string equivalence (as in the traditional matching) and RDF type equivalence (using URI string equivalence of the URIs used to identify the RDF type). In accordance to the RDF model, matching on RDF statements considers only the first three fields of the RDF tuple. The fourth field is used as an identifier for the RDF statement (s,p,o) and is therefore handled separately in semantic matching operations. A match on the ID field occurs only if both URIs are exactly the same. The complete match functions like a Boolean AND operation, i.e. the RDF matching on the first three fields AND the ID matching on the fourth fields must both return true. Generally, in a RDF matching the template will use the wildcard for the fourth field. Hence, in the case of the proposed Linda primitives for the RDF syntactic matching (inr, rdr) if the fourth field is not stated, it is automatically interpreted as the wildcard.

However in this case RDF has a number of special syntactic constructs which need to be handled specifically in matching RDFTuples. These constructs were already discussed in Chapter 3, where we considered their representation within the tuplespace. Here we discuss them again but in the context of respecting the RDF syntax when performing syntactic matching.

Blank nodes are identified by the internal type ts:BlankNode. It makes no sense to try to match Blank node instances as instances have no externally valid identifier - the identifier returned when a Blank node is bound to a variable is system determined and unique only to the given matching. However Blank nodes will match on variables of type ts:BlankNode or of course wildcards.

Matches on reified triples, containers or collections only occur where the field variable type in the template will match the RDF class Statement, List, Bag, Seq or Alt, respectively. In the case of constants, collections and containers are treated as array datatypes in which the unordered Alt or Bag matches any other Alt or Bag containing *exactly* the same set of objects while the ordered List and Seq matches only with the same set of objects *in the same order*. Each individual object is matched according to its own type.

Matching RDF Statements is like an additional tuple matching, following the same rules as the matching of the containing tuple. Subject, predicate, object and ID must all match.

While this matching provides a retrieval mechanism which takes into account RDF types and the special syntactic constructs of blank nodes, containers, collections and reification, the semantics of the RDF is still not taken into account. If the full power of RDF is to be available, we need to also allow matching at the information view of the Space, which makes use of ontological information associated to the RDF statements in form of RDFS or OWL ontologies.

5.3 Matching RDF Semantics

In the RDF model-theoretic semantics, RDF resources are no longer considered as instances of a datatype, but at a higher, knowledge representation level they are interpreted as concepts which have some agreed-upon meaning. It is out of scope of this report to go into details on the theoretic semantics of RDF or to further discuss its relationship to the RDFS and OWL semantics. [3]

Matching of RDFTuples in the information view shall support the interpretation of tuple content according to the defined RDF semantics. These matchings may choose to support interpretations at different levels of expressibility, in order to provide a compromise with computational complexity (the more expressive the interpretation we attempt to match, the more computationally complex it will be). It is clear that such matching can only occur if the RDF Schema (or OWL) information is available, otherwise it is not possible to draw the necessary inferences and the matching will take place according to the syntactic matching of the inr/rdr primitives.

[3] An entire discussion of RDF Semantics is at http://www.w3.org/TR/rdf-mt/

As an example of a core matching algorithm for the information view of Semantic Web Spaces we consider a standard, and generally fundamental, semantic interpretation: that of subsumption. Subsumption functions on the basis of subClass and subProperty statements in the RDF Schema. Formally these statements are equivalent to the respective Description Logics syntax:

C1 subClassOf C2 : $C1 \sqsubseteq C2$
P1 subPropertyOf P2 : $P1 \sqsubseteq P2$

We also note that these properties are transitive, i.e. if C1 subClassOf C2 and C2 subClassOf C3 then C1 subClassOf C3. In such a matching, we extend matching on RDF types to consider if a type T1, in the template, is subsumable by a type T2, in a tuple. Subsumption checking can be handled by any Description Logic reasoner. Hence for any T1 in a template, it is considered a match with any T2 in a tuple iff $T1 \sqsubseteq T2$. [4]

This also affects the matching on RDF constructs. Blank nodes now also match on variables typed as RDF resources (as we define ts:BlankNode as a subclass of rdf:Resource in the tuplespace ontology) and RDF statements will match on variables typed as RDF tuples (as we define rdf:Statement as a subclass of ts:RDFTuple in the tuplespace ontology).

This matching should also be further extendable, e.g. to support OWL based inferences provided OWL information is available and an OWL reasoner.

5.4 Template syntax

We now turn to the description of the template language in Semantic Web Spaces.

We consider a template language for constructing queries on the tuple space. A normal template, as in Linda, models the equivalence of a “simple query” in RDF query languages. For example (we use here a abbreviated syntax), a RDF query like

```
SELECT ?x WHERE (book, title, ?x)
```

is equally expressible as

```
endorse(book, title, ?x)
```

Note that RDF queries typically return variable bindings (where all variables in the query must not be bound in the response) while in the Semantic Web Space a Subspace with a matched Tuple (or set of Tuples) is returned (where no variable in the template is bound).

Rather than extend this simple template syntax to support more complex RDF queries, we consider that this simple template form retrieves a subset of a matching RDF graph

[4] One could also imagine further variations of this basic form of semantic matching, in which for example only direct ancestor or descendant types T2 are considered to match T1, in contrast to the case in which the complete transitive closure implied by the subClassOf relationship matches the given type T1.

(or many subsets, in the case of excerpt) which can then be further processed by the client using a dedicated RDF reasoner and its respective RDF query syntax. This is more efficient than attempting to perform complex RDF querying over the Semantic Web Space on the side of the space.

6 Tuplespace ontology

The structure of Semantic Web Spaces is represented explicitly by means of a so-called tuplespace ontology. The ontology describes the typical components of the tuplespace, such as sub-spaces, supported tuple types and matching templates, and coordinates the information access. The ontology provides an explicit means to model access rights, trust policies and active contexts of clients (here named agents) of the space. The internal representation of access rights and trust policies is currently undefined. It can easily be extended with new types of tuplespaces, rules describing new primitives and further metadata about spaces, and allows the usage of automatic reasoning in the management of tuples and accessing agents. The ontological model of the tuplespace references the concrete spaces and triples published by different parties, which are syntactically RDF documents and (sets of) RDF statements, respectively. Representing the entire tuplespace as an ontology model offers a means to reference tuples and tuplespaces which can be identified and addressed using URIs – by definition assigned to named Semantic Web resources.

A part of the tuplespace ontology is shown below in Figure 2.

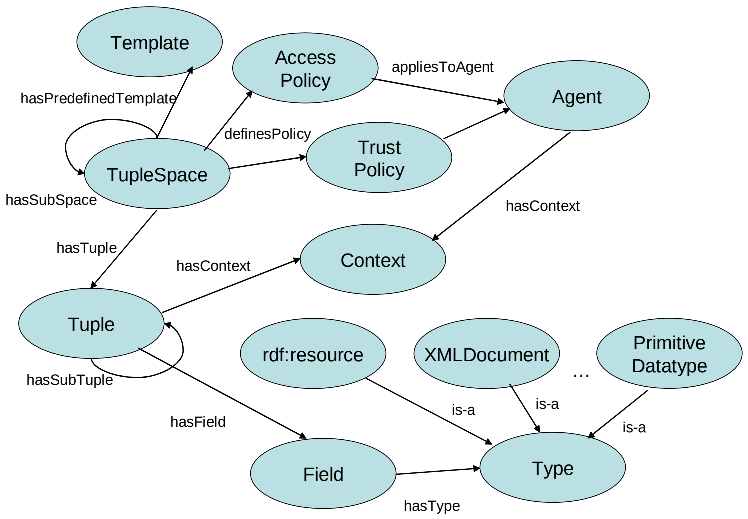

Figure 2: Ontology for a Semantic Web Space

Some examples will illustrate, in terms of RDF/RDFTuples, how particular knowledge can be expressed through the tuplespace ontology and subsequently extracted by a process using the Semantic Web Space.

6.1 Giving an ID to a tuple

In the Semantic Web Space, RDF tuples have been defined as being of the form <s, p, o, id>. However in the traditional RDF model, RDF statements are triples, i.e. they are of the form <subject, predicate, object>. A design decision of Semantic Web Spaces was to provide the tuples with IDs so that each could be uniquely identified. We have already noted that this is not the same as the reification of RDF statements. Hence, the question arises of how we model the tuple IDs in the space. The solution is that we define Tuples as first class objects within the tuplespace ontology, which means that they can be given an RDF ID. Every RDF tuple which is inserted into the space (whether by outr or by claim) generates a ts:RDFTuple instance (any tuple inserted by the standard out operation is a ts:Tuple instance, of which RDFTuple is a subclass). The space allocates internally IDs to every new Tuple instance, e.g.

```
<ts:RDFTuple
  rdf:about="http://nbi.inf.fu-berlin.de/sws/rdf/00823745">
.....
</ts:RDFTuple>
```

The ID URIs are recommended to follow W3C guidelines, i.e. to be built from an unique URI which is maintained by the Semantic Web Space host (e.g. a space hosted by the Computer Science department of the Free University of Berlin could take the URI sws.inf.fu-berlin.de). Such URIs should also be persistent over time, even if the tuple contents are not.

6.2 Getting the ID of a tuple field or field value

Tuples are also related to their tuple fields within the tuplespace ontology so that a knowledge base built from the ontology can represent the entire space, including all of its contained tuples. A standard Tuple has the property hasField, which points to a field instance containing the field value which is typed by the field type. RDFTuples have the subproperties hasSubject, hasPredicate and hasObject. There are cases where retrieving the field ID may prove useful, e.g. to reference a blank node (which has no identifier of its own, but in the tuplespace exists as the subject or object of a particular tuple) or the field value ID, e.g. referencing a RDF container/collection (which is represented by an array in the tuplespace).

In the first case, the tuplespace knowledge base could contain a RDFTuple like this:

```
<ts:RDFTuple rdf:about="http://nbi.inf.fu-berlin.de/sws/
            rdf/00453625">
 <ts:hasSubject>
  <ts:Field rdf:about="http://nbi.inf.fu-berlin.de/sws/
```

```
                rdf/00453625/1">
      <ts:hasValue>
       <book:Author
               rdf:about="http://www.robert-tolksdorf.de"/>
      </ts:hasValue>
     </ts:Field>
    </ts:hasSubject>
    <ts:hasPredicate>
     <ts:Field rdf:about="http://nbi.inf.fu-berlin.de/sws/
                rdf/00453625/2">
      <ts:hasValue>
       <rdf:Property
        rdf:about="http://www.books.com/ont#has-authored"/>
      </ts:hasValue>
     </ts:Field>
    <ts:hasPredicate>
    <ts:hasObject>
     <ts:Field rdf:about="http://nbi.inf.fu-berlin.de/sws/
                rdf/00453625/3">
      <ts:hasValue>
       <ts:BlankNode rdf:ID="75ge342" />
      </ts:hasValue>
     </ts:Field>
    </ts:hasObject>
   </ts:RDFTuple>
```

Note that the object of the RDFTuple is typed as a Blank node and the ID given is purely internal, i.e. it is not available to any agent retrieving this tuple. A possible approach to enabling a reference to a blank node (or, better said, a field containing a blank node) which is valid for the entire tuplespace would be to retrieve the ID of the field whose value is of type ts:BlankNode.

In the second case, the tuplespace knowledge base could contain a RDFTuple like this (the predicate is omitted for brevity):

```
   <ts:RDFTuple rdf:about="http://nbi.inf.fu-berlin.de/sws/
                 rdf/00823745">
    <ts:hasSubject>
     <ts:Field rdf:about="http://nbi.inf.fu-berlin.de/sws/
                rdf/00823745/1">
      <ts:hasValue>
       <book:Author
               rdf:about="http://www.robert-tolksdorf.de"/>
      </ts:hasValue>
     </ts:Field>
    </ts:hasSubject>
    ...
    <ts:hasObject>
     <ts:Field rdf:about="http://nbi.inf.fu-berlin.de/sws/
                rdf/00823745/3">
      <ts:hasValue>
       <rdf:List rdf:about="http://nbi.inf.fu-berlin.de/sws/
                    rdf/00823745/3/list">
        ...
       </rdf:List>
      </ts:hasValue>
     </ts:Field>
    </ts:hasObject>
   </ts:RDFTuple>
```

The object of this RDFTuple is a RDF List, which is stored at the tuple level (as opposed to the tuplespace level) in the form of an array of integers. If this tuple is retrieved by an agent, the object of the tuple is passed as a single array, however the object is typed in the tuple as a RDF List and this is made explicit in the tuplespace ontology. At the tuple level then the List has no referencable URI, rather, one could use the below templates to acquire the ID (where x is the instance of ts:Field returned from the first operation):

```
endorse("http://nbi.inf.fu-berlin.de/sws/rdf/00823745"
    [ts:RDFTuple], ts:hasObject, ? [ts:Field]),
endorse( x [ts:Field],ts:hasValue, ? [rdf:List]).
```

On this basis, further operations could be performed on the List (though we recommend retrieving the array object and performing operations client side if possible). For example, membership can be checked in the information view, on the basis that a field value which is typed as a RDF List (or as a container) generates implicit (or inferrable) tuples that associate each member of the RDF container/collection to the field instance through the relevant membership property.

```
endorse("http://nbi.inf.fu-berlin.de/sws/rdf/
    00823745/3/list"[rdf:List], rdfs:member, ? )
```

6.3 Checking the scopes of a tuple and adding them to an agent

The conceptual model also proposes that tuples and agents alike exist within certain contexts, otherwise known as scopes. These act to partition the tuples existence in or the agents view upon the tuplespace, respectively. One requirement that arises with the use of scopes is the need for agents to be able to check which scopes a tuple exists in, e.g. that once a tuple has been retrieved because it exists in one of the agents scopes, that the agent could check which other scopes the tuple also exists within and attempt to join those scopes. The tuplespace ontology provides a hasScope property upon tuples and agents which takes Scope instances as values. The scopes are identified by URIs as RDF IDs within the tuplespace ontology. Hence, once an agent has the ID of a Tuple it could use the following primitives to retrieve scopes and add them to its scope. The latter operation could return false, when rules also exist within the tuplespace which constrain which scopes an agent could belong to. Hence the semantics of the claim primitive are maintained (returning false when a claim is unsatisfiable) but this presupposes a higher level of reasoning than currently proposed for the RDFSpace.

```
excerpt("http://nbi.inf.fu-berlin.de/sws/rdf/00823745"
    [ts:RDFTuple], ts:hasScope, ? [ts:Scope]),
```

then for each Scope instance x,

```
claim("http://nbi.inf.fu-berlin.de/agent/114"[ts:Agent],
    ts:hasScope, x[ts:Scope])
```

6.4 Comparing access policies of a tuplespace and an agent

Finally, the tuplespace ontology also allows for the maintenance of policies relating to security and trust issues. These policies are generally associated to a tuplespace and to individual agents, and are used to determine whether a given agent can be trusted (i.e. that claims made in the space are trustworthy) or if certain actions of an agent are to be permitted (i.e. that the agent has access to that functionality in the space). In the latter case, it may be that certain agents may only access the data view of the space, or only non-destructively read tuples, or are restricted to a certain set of scopes which they may not change. How these policies may actually be represented is not defined in the conceptual model, it may be that they point by URL to a separate file using a specific syntax, or that the policies are themselves represented within RDF syntax (although they may be represented in a higher level logic, e.g. OWL or Rules. Approaches using Semantic Web languages to represent this kind of policies are already available in the literature). For this example we restrict ourselves to handling the policies as instances of the classes ts:AccessPolicy or ts:TrustPolicy, without making any further determinations on how those instances may relate further to the actual policies (as URLs, inline RDF etc.). Given the identification and possible authentication of the agent, a policy controller seeks to compare the access policy of the space with that of the agent:

```
endorse("http://sws.inf.fu-berlin.de"[ts:TupleSpace],
ts:definesPolicy, ? [ts:AccessPolicy]),
endorse( ? [ts:AccessPolicy],ts:appliesToAgent,
"http://nbi.inf.fu-berlin.de/agent/114"[ts:Agent])
```

7 Conclusions and Future Work

This technical report has outlined the conceptual model of Semantic Web Spaces in sufficient detail for a concrete implementation.

We have examined how Linda and tuplespaces can be used as a middleware technology for the Semantic Web, allowing us to benefit from their main properties: powerful coordination capabilities, asynchronous messaging and uncoupling of processes from space and time. We have also shown that to make this integration, certain changes in the tuple structure, co-ordination model and matching rules need to be made. We outlined what we foresee these changes to be. We also presented an ontology which is able to represent the tuplespace in an explicit, formal way, thus allowing reasoning over the management of tuples or clients. The proposed tuplespace architecture, based too on the interpretation of a defining ontology, is inherently extendible, so that support for further requirements of a real world Semantic Web middleware can be built in.

In a complementary technical report [TNPB05a], we will outline an use case for Semantic Web Spaces and illustrate the conceptual model described in this document through some concrete examples.

References

[Gel85] David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[HM04] Patrick Hayes and Brian McBride. RDF Semantics. *Available at http://www.w3.org/TR/rdf-mt/*, 2004.

[HPSB$^+$04] 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.

[JF04] B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. *Journal of Systems and Software*, 69(3):243–266, 2004.

[MW00] Iain Merrick and Alan Wood. Coordination with Scopes. In Janice Carroll, Ernesto Damiani, Hisham Haddad, and Dave Oppenheim, editors, *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC 2000)*, pages 210–217, Como (I), 19–21 March 2000. ACM. Track on Coordination Models, Languages and Applications.

[PSHH04] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. *Available at http://www.w3.org/TR/owl-absyn/*, 2004.

[RCD01] 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.

[RW98] A. I. T. Rowstron and A. M. Wood. Solving the Linda multiple rd problem using the copy-collect primitive. *Sci. Comput. Program.*, 31(2-3):335–358, 1998.

[TNL$^+$04] 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.

[TNPB05a] R. Tolksdorf, L. Nixon, and E. Paslaru Bontas. Using Semantic Web Spaces to realize Ontology Repositories (Technical Report TR-B-05-15). Technical report, Free University of Berlin, 2005.

[TNPB^{+}05b] R. Tolksdorf, L. Nixon, E. Paslaru Bontas, D. M. Nguyen, and F. Liebsch. Enabling real world Semantic Web applications through a coordination middleware. In *Proceedings of the 2nd European Conference on Semantic Web ESWC2005*. Springer Verlag, 2005.

[TPBN05] R. Tolksdorf, E. Paslaru Bontas, and L. Nixon. Towards a tuplespace-based middleware for the Semantic Web. In *Proceedings of the IEEE Web Intelligence Conference WI2005*, 2005.