Data Mediation Support for Triple Space Computing

Omair Shafiq[1], François Scharffe[1], Reto Krummenacher[1],
Ying Ding[1,2], Dieter Fensel[1,2]

[1] Digital Enterprise Research Institute (DERI)
University of Innsbruck (UIBK)
6020 Innsbruck, Austria.

[2] Electronic WebService GmbH (eWS)
6020 Innsbruck, Austria.

Email: {omair.shafiq,francois.scharffe,reto.krummenacher,
ying.ding,dieter.fensel}@deri.org

Abstract

Triple Space Computing is an emerging technology for communication and coordination of different semantic technologies. It has been achieved by extending Tuple Space computing to support RDF as Triple Space Computing. It can play an important role by acting as global middleware providing support for collaboration of different semantic technologies. Several participants from Semantic Web and Semantic Web Services applications, distributed worldwide communicate with each other via triple space due to which the possibility of heterogeneity in the RDF schemas and instances among the communicating participants may arise. In order to resolve this heterogeneity issue, data mediation becomes an important part of the Triple Space Computing architecture. In this paper we introduce data mediation support for Triple Space Computing and further explain how mismatches occur in RDF data, presents an Abstract Mapping Language to specify mapping to cover the heterogeneity, defines architecture of the mediation engine and its interfaces, processing of mediation mapping rules, grounding of mediation rules as RDF triples and finally evaluates with examples.

1. Introduction

Triple Space Computing [9] is an emerging technology as extended Tuple Space Computing to support Semantic Web [1] technologies RDF and enables asynchronous communication among Semantic Web Services based on persistent publication and read of RDF triples. It provides a shared space to multiple participants communicating with each other. However, due to the diversity in the nature of different communicating participants, the possibility of the heterogeneity in metadata (RDF schema) of different participants arise which makes mediation an important issue to be resolved.

Mediation in Triple Space Computing is concerned with handling heterogeneity by resolving possibly occurring mismatches among different triples. There can be a possibility that different TSC participants communicating via triple spaces, containing different data models which makes mediation an important issue to be taken care of. So, a RDF instance in a RDF schema of one TSC participant is needed to be represented in the RDF schema of the other TSC participant without altering or loosing the semantics. For this reason, a mapping language is needed that specifies how to trans-form RDF triples according to different RDF Schemas of different participants. The mediation rules are to be specified at design time which will be processed by a mediation engine in the TSC framework at runtime in order to carry out the mediation during the communication among TSC participants.

The paper is structured as follows: it introduces Triple Space Computing for Semantic Web Services in section 2. Triple Space Kernel as realization of Triple Space Computing paradigm is described in section 3. In section 4, we present an abstract mapping language and its use in TSC data mediation, specifies mediation mapping rules in the abstract mapping language, specifies mediation API interface, architecture and working of mediation engine in TS kernel and grounding of mediation mapping rules as RDF triples. Section 5 evaluates with examples and finally draws conclusion.

2. Triple Space Computing for Semantic Web Services

Semantic Web Services promise seamless interoperability of data and applications on a semantic level, thus turning the Web from a world-wide information repository for human consumption only to a device of distributed computation. To this end, appropriate semantic descriptions of Web services and intelligent mechanisms working upon this need a solid basement in terms of the underlying semantically enabled communication technologies. Here, Triple Space Computing (TSC) comes into play which defines the technologies and settings needed to develop a new paradigm for Web service communication that complies with the basic principles of the Web, i.e. state-less communication of resources, persistent publication of resources, unique identification of resources and non-destructive read access to resources.

The basic foundation here is Tuple Space computing. The tuple space model not only decouples the three orthogonal dimensions involved in information exchange: reference, time, and space, but also offers a high-level abstraction, namely the communication via reading and writing data in a space. This offers the advantage of re-moving the complexity of message exchange based systems and of the APIs currently used for building Web services. It also offers advantages in terms of reduced development costs, simplicity, extensibility, expressiveness, easy debugging, scalability, failure tolerance and recovery. This can benefit the construction, analysis, testing and reusability of distributed applications.

Triple Space Computing further adds compatibility with Web design principles, thus overcoming the deficiencies of message-based communication. Therefore, it adds some additional features that are currently lacking in the tuple space paradigm, i.e. URIs as a unique and well-defined reference mechanism that has proven suitability on Web scale, namespaces as a separation mechanism for distinguishing chunks of information by qualified names, publication-based communication paradigm that scales because of its simple and universal dissemination manner, interlinking of resources by use of foreign URIs as hyperlinks.

Triple Space Computing will be used as communication paradigm for Semantic Web Services [18] and may also be for other similar technologies like Semantic Grid Services [14]. It improves the current communication paradigm of Web services by providing asynchronous invocation support. It acts as third party among the Web service clients and Web services. When a Web service client sends a request to some Web service, it should publish data in the Triple Space Computing middleware. Similarly, Web service can receive the invocation request from client by reading data from the Triple Space middleware. The same applies to semantically described Web services as Semantic Web Services. We started to investigate that how the Triple Space Computing could be applied to our implementation for Semantic Web Services i.e., The Web Service Execution Environment (WSMX) [17] which is a reference implementation of Web Services Modeling Ontology (WSMO) [2]. The use of

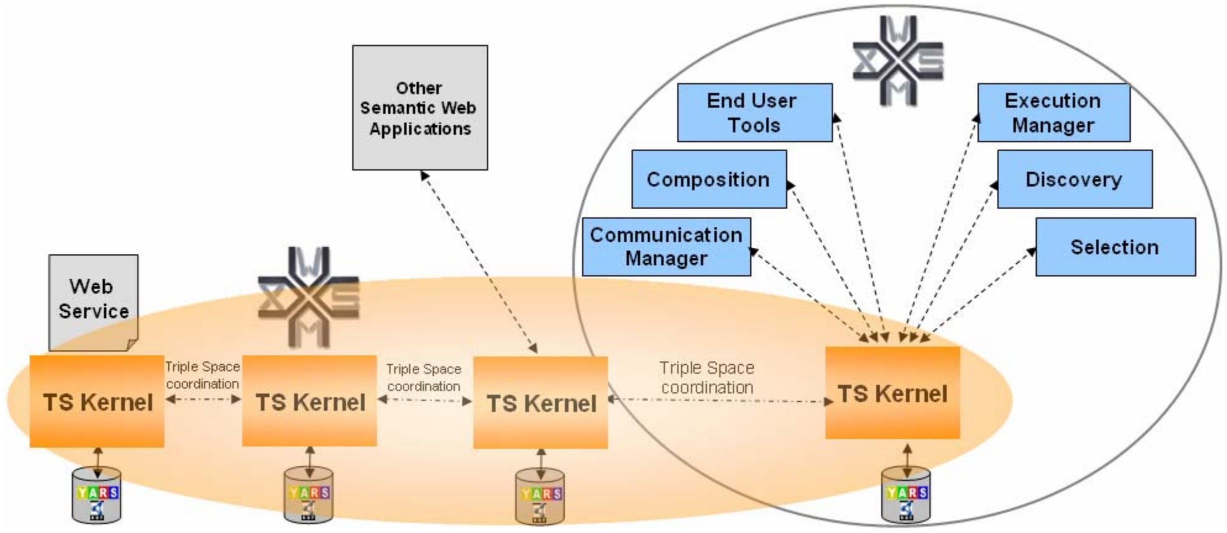

Figure 1: Triple Space Computing as collaborative global middleware

Triple Spaces for asynchronous communication between different WSMXs brings them a step closer to their architectural goal, i.e. to support greater modularization, flexibility and decoupling in communication of different WSMX nodes. Similarly, it enables WSMX to be highly distributed and easily accessible. Furthermore, being a third party element Triple Space Computing can resolve communication disputes that may arise.

3. Triple Space Kernel

The Triple Space kernel (TS kernel) is the concrete realization of Triple Space Computing paradigm which can be used to implement both Triple Space servers and heavy clients. In the former case it also provides a proxy component, which allows light clients to remotely access the server. The TS kernel itself consists of multiple components, i.e. operations and security layer, mediation engine, coordination and data access layer. Figure 2 shows the abstract architecture and a brief overview is given below.

The operations and security layer executes Triple Space operations issued by participants. Heavy clients run in the same address space as the TS kernel, and the TS kernel is accessed by its native interface. Light clients use TS proxies to access the TS kernel of a server node transparently over the network. As a variation a light client can access a TS kernel via a standardized protocol, e.g. HTTP. In this case a server side component, e.g. a servlet, translates the protocol to the native TS kernel interface. The execution of a TS operation includes verification of security constraints, maintaining state of blocking operations and invocation of the underlying coordination layer. The security management API is used to define and change security configurations [19] such as access control for spaces or named graphs [20].

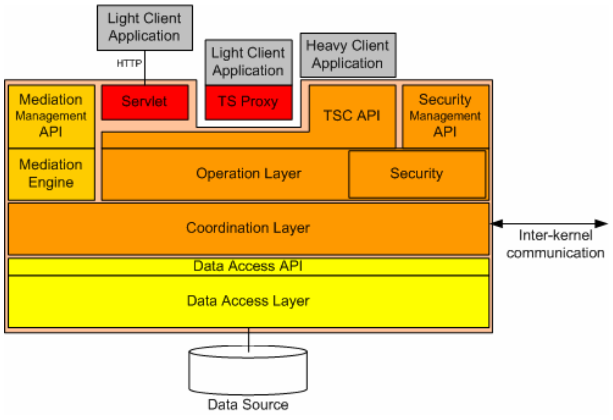

Figure 2: Triple Space Kernel

The mediation engine resolves heterogeneity issues by providing mappings for possibly occurring mismatches among different RDF triples. It is due to the possibility that different participants may have different RDF schemas while communicating via triple space. Mapping rules for mediation are provided to the mediation engine at design time and are processed by it at run time in order to resolve heterogeneities by identifying mappings. The mediation management API provides methods to turn on/off the usage of mediation engine, to add, remove and replace mediation rules.

The coordination layer implements transaction management, i.e. the creation, commit and abort of a transaction and guarantees that concurrent operations are processed consistently. It accesses the local data access layer to retrieve data from a space and to apply permanent changes to a space. Further, if a Space is spanned by multiple TS kernels, the coordination layer is responsible for inter-kernel communication to distribute and collect data and to assure that all involved kernels have a consistent view to a space.

The data access layer acts as a gateway to the underlying data storage. It abstracts from the underlying details of storage systems. All data stores should be accessed through this layer. This layer provides generic interfaces for accessing different data sources. Therefore, this layer is responsible for providing accessibility to every data store used in TSC.

Figure 2 shows briefly that how different Triple Space kernels can coordinate with each other to make the Triple Space Computing as global communication and coordination middleware for effective collaboration of different technologies based on Semantic Web, Web Services and Semantic Web Services.

4. Mediation in TSC

This section explains the mediation process in TSC which introduces the specification and processing of mapping rules during the communication, i.e. from RDF schema of requesting participant to the RDF schema of the target participant. A mapping rule contains a link between rdfs:Resources from the source schema to the target schema. Figure 3 below shows the mediation process which is RDF schema level mediation. The mapping rule can also link different particular RDF instances, which will be instance level mediation. The mediation process in TSC will follow the approach as, the instance I of a RDF schema R would be mapped to the instance I′ of the RDF schema R′ using some mapping rule M_R. Where the mapping rule should be created for each pair of RDF schema at design time.

The mediation rules are to be specified in an abstract mapping language and to be provided at design time. It would be processed by the mediation engine to identify different possible heterogeneities at instance and schema level and carry out mediation according to the mediation rules at runtime during the communication of participants.

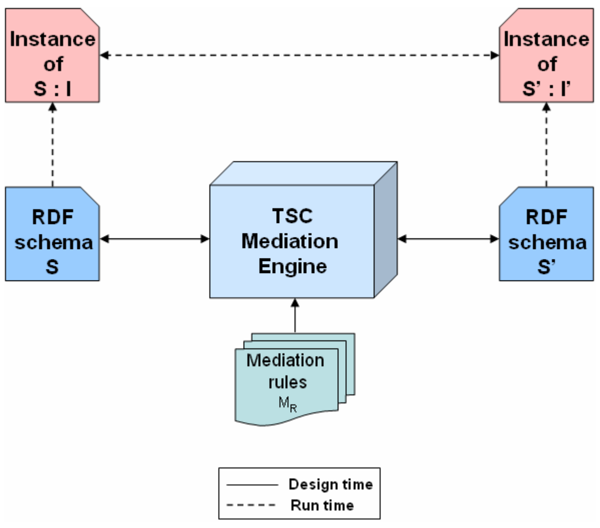

Figure 3: Mediation in Triple Space Computing

The mediation engine is used when a template is provided by a participant to match with RDF triples available in triple space storage. The mediation engine first of all extracts information about participant's desired resources and checks for mediation by processing the available mediation rules provided to it at design time. If any rule is found related to a resource mentioned in template, it means that the resource in template can be mapped to one or more resources available in the triple space storage as mentioned in the rule. As a result, separate templates will be generated for all the corresponding matched resources according to the mediation rule. All the new generated templates along with the original one will be matched during the search process by query engine and results will be returned to the user.

Mapping rules express the correspondence among different RDF schemas and instances. In this section, we describe the details of specification of the mediation rules in order to explicitly identify the relations that exist among resources of different RDF triples. The mapping rules are also proposed to be represented as RDF triples which could easily be stored on the same triple space storage instead of having a separate storage mechanism to store the rules.

4.1. Abstract Mapping Language

This section presents a brief overview of an abstract mapping language that allows expressing mediation mapping rules. It was developed jointly by Web Service Modeling Ontology Working Group (WSMO) [15] and Ontology Management Working Group (OMWG) [16]. It supports both schema and instance level mappings. It allows various types of mappings i.e. one to one, one to many, many to one and many to many mappings with the help of a set of operators that are introduced to link together different entities. It also contains constructs to condition the validity of the mappings.

The mapping language has abstract syntax expressed in Extended Backus-Naur Form (EBNF). It allows optional elements that could be mentioned between square brackets '[' and ']'. Elements between curly brackets '{' and '}' can have multiple occurrences. Each element of ontology (in this case, RDF schema), whether it is a class, attribute, instance, or relation, is identified using a IRI which is an extension of a URI. The language also allows concrete data values, i.e. type literals, plain literals, numeric values and strings. A mapping can be either unidirectional or bidirectional. A unidirectional mapping from a source to a target means that the source expression is subsumed by the target one. A bidirectional mapping means the source and the target expressions are equivalent. Simple mapping expressions are expressed by replacing the class, attribute and relation expressions with corresponding identifiers of the respective class attribute or relation.

A set of expressions from the abstract mapping language that can be used for mediation between RDF schema and instances are as follows:

```
**Mapping expression** = {logical_expression}
lbrace annotation* logicalexpression rbrace

| {class_mapping} classmapping lpar
annotation* measure? directionality? [first]:
classexpr [second]: classexpr classcondition*
logicalexprbrace? rpar

| {relation_mapping} relationmapping lpar
annotation* measure? directionality? [first]:
relationexpr [second]: relationexpr
relationcondition* logicalexprbrace? rpar

| {instance_mapping} lpar annotation* [first]:
instanceid [second]: instanceid rpar

Where:  annotation = t_annotation lpar
propertyid propertyvalue rpar
     measure  =    t_measure lpar float rpar
```

```
lpar  =  '(', rpar  =  ')', lbracket
=  '[', rbracket  =      ']', lbrace
=  '{', rbrace  =  '}'
```

4.2. Mediation rules specification in Abstract Mapping Language

Mapping rules in TSC will express the correspondence between different resources in different RDF triples. In this section, we describe the details of specification of the mediation rules in order to explicitly identify the relations that exist among resources among different RDF triples. The mapping rules are also proposed to be represented as RDF triples which could easily be stored in same Triple Space instead of having a separate storage mechanism to store the rules. In TSC, mappings are always supposed to be resource to resource mapping among different RDF triples. It can either be an instance level mapping or schema level mapping and there are further two options in each type of mapping in order to explicitly specify, whether it is a unidirectional or bidirectional.

1) resource A **is equivalent to** resource B
   It is a bidirectional mapping which means that resource A can be mapped to resource B and resource B can also be mapped to resource A
2) resource A **subsumes** resource B
   It is a unidirectional mapping which means that resource A can be mapped to resource B whereas resource B cannot be mapped to resource A

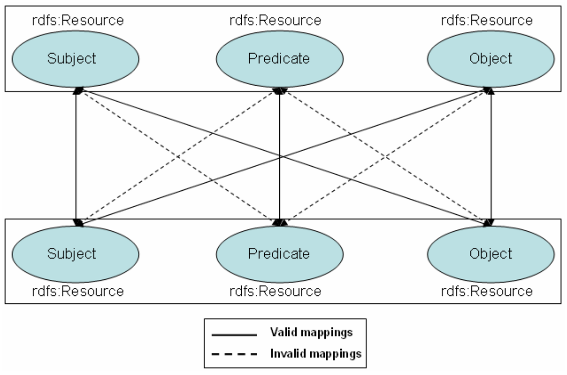

Figure 4: Mapping possibilities among RDF triples

While representing the mappings among the resources of RDF triples, there are certain constraints that exist which do not allow mappings between Subject and Predicate or Object and Predicate of the RDF triples. Figure 4 below explains all possibilities of mappings in a single snapshot.

- A subject or object can be mapped to a subject or object. Mapping will be carried out using rdfs:subClassOf, i.e. C1 rdfs:subClassOf C2
- A predicate can be mapped to a predicate. Mapping will be carried out using rdfs:subPropertyOf , i.e. P1 rdfs:subPropertyOf P2
- A subject or object cannot be mapped to a predicate and vice versa

Based on the assumptions mentioned above for data mediation in Triple Space Computing, the abstract mapping language can be used to specify the mappings expressions among classes, properties of RDF schema and instances. Given below are set of expressions along with their descriptions that allow specifying the mediation mapping rules. Table 1 below provides the set of expressions for mediation mapping rules along with a brief description.

Abstract Mapping Language	Description
`T( MappingDocument( A source_exp target_exp annotation1 ... annotationn expression) )`	Provides a start point to the document describing mapping expressions to bind the source and target schemas.
`T( classMapping( annotation1 ... annotationn directionality classExpr classExpr classCondition1 ... classConditionn logicalExpression) , A)`	Specifies mapping expressions between two classes, defines type of mapping (i.e. unidirectional or bidirectional), mapping conditions (i.e. one to one, one to many, many to one and many to many).
`T( relationMapping( annotation1 ... annotationn directionality relationExpr relationExpr relationCondition1 ... relationConditionn logicalExpression) , A)`	Specifies mapping expressions between two relations (properties), defines type of mapping (i.e. unidirectional or bidirectional), mapping conditions (i.e. one to one, one to many, many to one and many to many).
`T( instanceMapping( annotation1 ... annotationn instance1`	Specifies mapping between two instances of ontologies using class-

`instance2 , A)`	mapping, relation-mapping as mentioned above.

Table 1: Mapping rules in abstract mapping language

4.3. Architecture of TSC Mediation Engine

This section presents architecture of the mediation engine inside Triple Space kernel that helps query engine to find out all the mediation mappings before it actually starts template matching. The design and integration of the mediation engine has been carried out in such a way that it acts as plug-in and can be used or not as required. Figure 5 below shows an abstract architecture of the mediation engine for Triple Space kernel. A brief explanation of individual components is given below followed by a description of overall working.

The **mediation management interface** binds the mediation engine with user interface in order to allow participants to interact with the mediation manager to turn on or off the usage of the mediation engine before template matching and to add, replace or remove the mapping rules. Mediation manager inside the mediation engine receives and serializes the mapping rules in abstract mapping language as set of RDF triples on Triple Space. The **system interface** allows the operations layer to communicate with mediation engine in order to find out any corresponding RDFS resources using the mediation mapping rules before search for RDF triples starts. The **template parser** receives template provided by a template handler in operation layer and parses out all RDFS resources values in there. The **expression generator** pulls out the set of RDF triples (mappings rules in abstract mapping language serialized as set of RDF triples) and reformulates a mapping expression out of it. The **mediation mapper** is the core component of mediation engine and processes the mapping rules for all the RDFS resources from template provided by operation layer in order to find out any corresponding RDFS resources. The **template generator** receives a set of corresponding RDFS resources from mediation

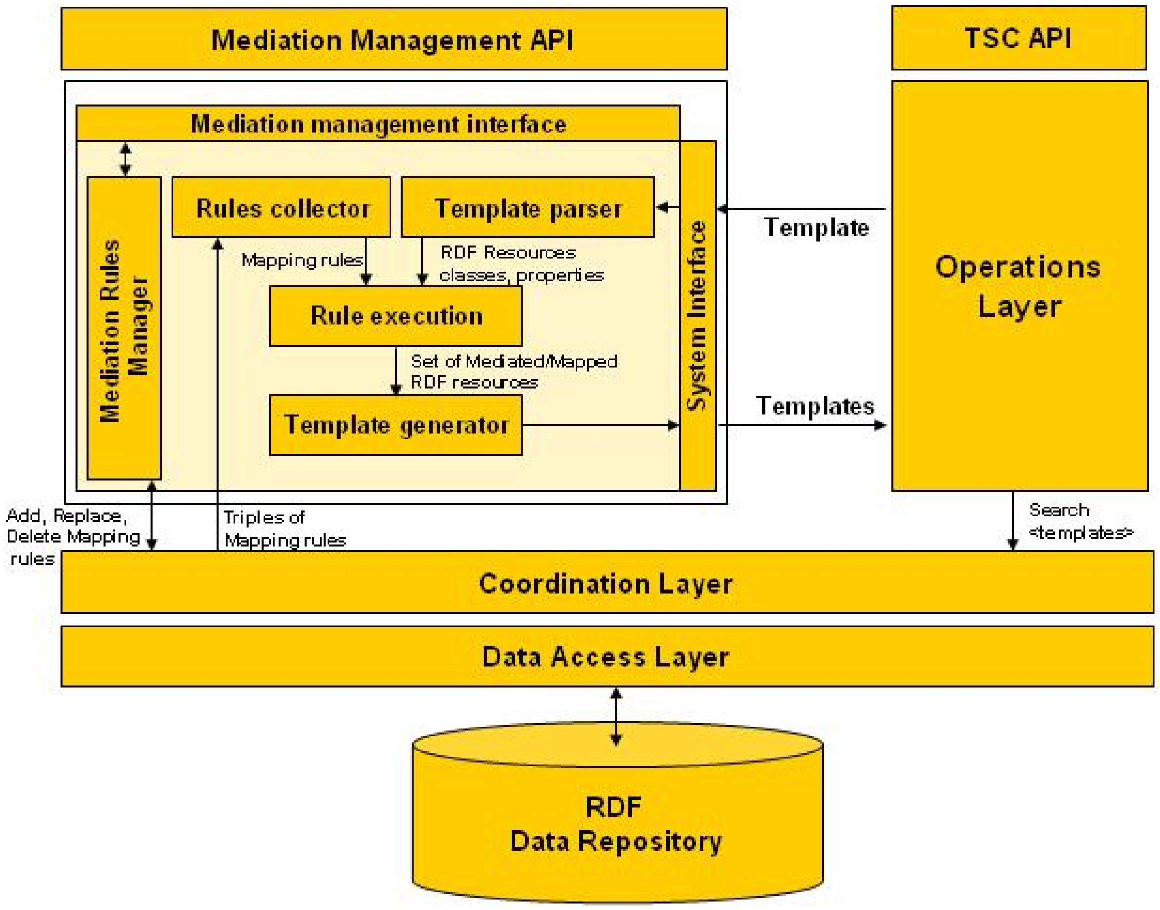

Figure 5: Architecture of the TSC mediation engine

mapper and generates multiple templates accordingly and finally forwards it to query engine to make a search in the triple space.

The working of TSC mediation engine starts when users (TSC participants) add mediation mapping rules via mediation management interface. The mediation manager takes care of serialization of mediation rules in the Triple Space storage as RDF triples. The mediation engine manager also helps adding, replacing and deleting the mapping rules. When a TSC participant wants to search something from the triple space, the operation layer generates a template which contains the information about required RDF triples and forwards it to the system interface of the mediation engine. The template parser in the mediation engine receives the template and parses out all the RDF schema resources mentioned in template and forwards it to mediation mapper. At the same time, the expression generator checks from triple space for any mediation mapping rules serialized as RDF triples and generates mapping expressions and forwards it to the mediation mapper. The mediation mapper being core of the TSC mediation engine executes the mapping expressions against the RDF schema resources (provided by template parser) and finds out any corresponding set of RDF schema resources. All the discovered set of RDF schema resources from mediation mapping rules are forwarded to template generator which encloses the RDF schema resources in templates and sends it back to operations layer. After all the above mentioned process, operations layer can forward the required triples to coordination layer and down to data access layer to search the required triples according the RDF schema resources mentioned by TSC participant along with the corresponding one found out by mediation process.

4.4. Mediation API interface specification

Two interfaces have been proposed in the TSC mediation engine. The mediation management interface allows users or TSC participants to access the mediation engine. Whereas, the system interface of the mediation engine allows operations layer in the Triple Space Kernel to interact with mediation engine to check for corresponding resources needed to be considered by query engine while searching in triple space.

The mediation mapping rules specified in the abstract mapping language can be added, removed or replaced from the TSC mediation engine using the mediation management interface. Moreover, as described above that mediation engine is an optional component in the TSC framework and can be turned on or off using mediation management interface. The table below provides a brief overview of operations in the interfaces of TSC mediation engine.

Operations	Description
Mediation management interface	
useMediationEngine (Boolean): void	Mediation Engine can be turned on or off as a plugin.
addRule(String rule): URI	A participant provides a mediation rule expressed using abstract mapping language
removeRule (URI): void	A participant removes a mediation rule by providing a URI to it.
replaceRule (String newRule, URI old): URI	A participant replaces a mediation rule. Basically composed of remove the old mediation rule using RemoveRule and adds new one using AddRule.
System interface	
mediationCheck (Template t): Set<Template>	Check for any mapping that exists for a particular rdfs:Resource in a triple.

Table 2: Mediation API interface

4.5. Grounding of mediation rules in RDF

As mentioned in while defining architecture of mediation engine, mappings rules are to be serialized as RDF triples to store them in Triple Space along with a URI to access it. Simple mappings can be realized as single triples along with a URI i.e. as quads whereas, complex mappings which could be based on multiple simple mappings can be stored as multiple quads forming a named graph. So, the mediation rules will look like quads where each quad having a source resource (the resource which is to be matched) represented as subject, mapping type (equivalent or subsume) represented as property and target resource (the resource with which source resource is to be mapped) represented as object. The forth part will be the URI to access this mapping rule. The mediation engine will contain a list of URIs of quads or named graphs to access the quads in the Triple Space having the mediation rules. The mapping rules will be stored and referred to by the mediation engine as (mediation specific) named graphs in the Triple Space.

For two different categories of possible mappings between the resources i.e. unidirectional and bidirectional (equivalent and subsume). The mappings mentioning equivalence of the two resources will be serialized as two quads having same URIs in order to explicitly mention the bidirectional nature of mapping among the resources. Whereas, a subsume mapping rule will be stored as a single quad mentioning the unidirectional nature of mapping among the resources, i.e. shown in the examples section below.

Based on the requirements and assumptions made above, the mediation mapping rules in abstract mapping language can be grounded into RDF triples as shown in the table below.

Mapping rules in AML	**Corresponding RDF triples**

`T(`
`MappingDocument(`
`A`
`source_exp`
`target_exp`
`annotation1 ...`
`annotationn`
`expression)`
`)`
	`A rdf#type`
	`map#mappingDocument`
	`T(source_exp,A)`
	`T(target_exp,A)`
	`T(annotation1,A) ...`
	`T(annotationn,A)`
	`T(expression`
	`, A)`

`T(`
`classMapping(`
`annotation1 ...`
`annotationn`
`directionality`
`classExpr`
`classExpr`
`classCondition1 ...`
`classConditionn`
`logicalExpression)`
`, A)`
	`A map#classMapping _:X`
	`T(annotation1, _:X)`
	`...`
	`T(annotationn, _:X)`
	`T(directionality, _:X)`
	`_:X map#hasSource _:Y`
	`_:X map#hasTarget _:Z`
	`T(classExpr, _:Y)`
	`T(classExpr, _:Z)`
	`T(classCondition1,`
	`_:X)`
	`...`
	`T(classConditionn,`
	`_:X)`
	`T(logicalExpression,`
	`_:X)`

`T(`
`relationMapping(`
`annotation1 ...`
`annotationn`
`directionality`
`relationExpr`
`relationExpr`
`relationCondition1 ...`
`relationConditionn`
`logicalExpression)`
`, A)`
	`A map#relationMapping`
	`_:X`
	`T(annotation1, _:X)`
	`...`
	`T(annotationn, _:X)`
	`T(directionality, _:X)`
	`_:X map#hasSource _:Y`
	`_:X map#hasTarget _:Z`
	`T(relationExpr, _:Y)`
	`T(relationExpr, _:Z)`
	`T(relationCondition1,`
	`_:X)`
	`...`
	`T(relationConditionn,`
	`_:X)`
	`T(logicalExpression,`
	`_:X)`

`T(`
`instanceMapping(`
`annotation1 ...`
`annotationn`
`instance1`
`instance2`
`, A)`
	`A`
	`map#individualMapping`
	`_:X`
	`T(annotation1, _:X)`
	`...`
	`T(annotationn, _:X)`
	`_:X map:hasSource`
	`instance1`
	`_:X map:hasTarget`
	`instance2`

Table 3: Mapping rules in abstract mapping language and their corresponding RDF triples

5. Examples

This section evaluates the mediation engine in Triple Space kernel with some real life examples by presenting different types of possible mismatches among RDF triples in the triple space storage and mappings that are needed to be specified in order to overcome the mismatches. Two sets of RDF triples are shown below in Table 4 about same information but represented in different ways:

- the project in first set of triples is mentioned as TSC whereas it is mentioned as Triple Space Computing in the second set of triples
- hasDeliverable property in first set of triples is mentioned as to hasProjectOutput property in the second set of triples
- doap:Project object in the first set of triples is mentioned as :ResearchProject

```
First set of Triples

<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns"
xmlns:project="http://www.deri.org/ns">
  <rdf:Description
rdf:about="http://www.deri.org/tsc">

<project:hasDeliverable>D1.3</project:hasDeliv
erable>

<rdf:type>"http://www.projects.org/doapProject
"</rdf:type>
  </rdf:Description>
</rdf:RDF>

----------------------------------------------

Second set of Triples
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
xmlns:research="http://www.tuwien.ac.at/ns">
  <rdf:Description
rdf:about="http://www.tuwien.ac.at/triple_spac
e_computing">

<research:hasProjectOutput>D1.3</research:hasP
rojectOutput>

<rdf:type>http://www.projects.org/researchProj
ect</rdf:type>
  </rdf:Description>
</rdf:RDF>
```

Table 4: Two sets of RDF triples requiring mediation

The following mappings have to be defined and provided to the mediation engine before hand in order to resolve the above mentioned mismatches before searching the triple space, i.e.

1) TSC is **equivalent** to Triple Space Computing
2) hasDeliverable is **equivalent** to hasProjectOutput
3) doap:Project **subsumes** :ResearchProject

The first mapping is an instance level mapping as it is concerned with mapping particular values of different RDF triples. The mediation mapping rule in abstract mapping language will be as follows:

```
T ( instanceMapping ( annotation ( propertyid
propertyvalue ) bidirectional “TSC” “Triple
Space Computing” )
```

The second mapping is a schema level mapping that is concerned with mapping two different properties defined in two different RDF schemas. The mediation mapping rule in abstract mapping language will be as follows:

```
T ( relationMapping ( annotation ( propertyid
propertyvalue          )          bidirectional
namespace:hasDeliverable
namespace:hasProjectOutput )
```

The third mapping is also an RDF schema level mapping as it is concerned with the mapping of two different set of classes defined in different RDF schemas. The mediation mapping rule in Abstract Mapping Language will be as follows:

```
T ( classMapping ( annotation ( propertyid
propertyvalue ) unidirectional doap:Project
:researchProject )
```

The above mentioned mapping rules in Abstract Mapping Language will be serialized as set of RDF triples, given below is the RDF representation of the above mentioned mediation mapping rules:

```
<?xml version="1.0"?>

Set of triples for first mapping rule

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
xmlns:project="http://www.deri.org/ns#">
  <rdf:Description
rdf:about="http://www.deri.at/projects/TSC">

<rdf:subClassOf>http://www.tuwien.ac.at/projec
ts/
     Triple_Space_Computing</rdf:subClassOf>
  </rdf:Description>
  <rdf:Description
rdf:about="http://www.tuwien.ac.at/projects/
     Triple_Space_Computing">

<rdf:subClassOf>http://www.deri.at/projects/TS
C</rdf:subClassOf>
  </rdf:Description>
</rdf:RDF>

----------------------------------------------

Set of triples for second mapping rule

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
xmlns:project="http://www.deri.org/ns#">
  <rdf:Description rdf:about="hasDeliverable">

<rdf:subClassOf>hasProjectOutput</rdf:subClass
Of>
  </rdf:Description>
  <rdf:Description
rdf:about="hasProjectOutput">

<rdf:subClassOf>hasDeliverable</rdf:subClassOf
>
  </rdf:Description>
</rdf:RDF>

----------------------------------------------

Set of triples for third mapping rule
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
xmlns:project="http://www.deri.org/ns#">
  <rdf:Description
rdf:about=":ResearchProject">

<rdf:subClassOf>doap:Project</rdf:subClassOf>
  </rdf:Description>
</rdf:RDF>
```

Table 5: Mappings rules represented as RDF triples

6. Conclusions and Future Work

Integration of data, information, knowledge, processes, applications, and businesses is one of the core problems in systems construction due to the heterogeneity of the interacting entities. Hence, Mediation is required in a heterogeneous environment like the Triple Space to overcome its heterogeneity. Mediation support in the system has been provided for a variety of purposes, i.e. to remove differences in the syntactic representation and the intended semantics of data that is exchanged or to remove differences in the way that different participants communicate with each other. In this paper we propose data mediation support in Triple Space Computing by providing a data mediation engine in Triple Space Kernel to cope with heterogeneity issue arises due to the difference in the RDF schema and instance. Future directions of the research work is to provide support for automated creation of mediation mapping rules by using some natural language processing and reasoning techniques.

Acknowledgements

The work is funded by the European Commission under the projects ASG, DIP, enIRaF, InfraWebs, Knowledge Web, Musing, Salero, SEKT, Seemp, SemanticGOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/I13 ; by the FFG (Österreichische Forschungsförderungsgeselleschaft mbH) under the projects Grisino, RW², SemNetMan, SeNSE, TSC and OnTourism.

References

[1] Berners-Lee, T., et. al: The Semantic Web. Scientific American (2001).

[2] Roman, D., Lausen, H., U.Keller, eds.: Web Service Modelling Ontoloty (WSMO). WSMO Deliverable, version 1.2 (2005)

[3] Martin, D., et. al: Bringing Semantics to Web Services: The OWL-S Approach. In Proceed-ings of the First International Workshop on Semantic Web Services and Web Process Com-position (2004)

[4] Patil, A., Oundhakar, S., Verma, K.: METEOR-S:Web Service Annotation Framework. In Proceedings of World Wide Web Conference (2004)

[5] Engleberg, I., Wynn, D.: Working in groups: Communication Principles and Strategies. Houghton Mifflin (2003)

[6] Gelernter, D.: Mirror Worlds. Oxford University Press (1991)

[7] TSpace, http://www.icc3.com/ec/tspace

[8] Eugster, P., et. al: The Many Faces of Publish/Subscribe. ACM Computing Surveys (2003)

[9] Fensel, D.: Triple Space Computing. Technical Report, Digital Enterprise Research Institute (DERI) (2004)

[10] Bussler, C.: A Minimal Triple Space Computing Architecture. In Proceedings of WIW 2005.

[11] Yet another RDF Store, http://sw.deri.org/2004/06/yars/yars.html

[12] J. Domingue, D. Roman, M. Stollber, “Web Service Modeling Ontology (WSMO) - An Ontology for Semantic Web Services”, Position paper at the W3C Workshop on Frame-works for Semantics in Web Services, June 9-10, 2005, Innsbruck, Austria.

[13] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel: WWW or What Is Wrong with Web Services. In Proc. of the 2005 IEEE European Conf on Web Services (ECOWS 2005), Växjö, Sweden, November 14-16, 2005.

[14] O. Shafiq, I. Toma, R. Krummenacher, T. Strang, D. Fensel, "Using Triple-Space comput-ing for communication and coordination in Semantic Grid”, 3rd Semantic Grid workshop at 16th Global Grid Forum (GGF), Athens Greece, February 13-16 2006.

[15] D. Roman, U. Keller, H. Lausen, J. de-Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel: Web Service Modeling Ontology, Applied Ontology, 1(1): 77 - 106, 2005.

[16] F. Scharffe, A. Kiryakov, OMWG - Mapping and Merging Tool Design, DERI OMWG Working Draft, October 2005. Available at http://www.omwg.org/TR/d7/d7.2/v0.2

[17] C. Bussler et al, Web Service Execution Environment (WSMX), W3C Member Submis-sion, June 2005. Available at http://www.w3.org/Submission/WSMX

[18] F. Martin-Recuerda and B. Sapkota (eds.). WSMX Triple-Space Computing. Deliverable D21, 2005; available at: http://www.wsmo.org/TR/d21

[19] M. Murth, J. Riemer, "Security and Privacy Models in Triple Space", a Final Working Draft of Austrian FIT-IT funded Triple Space Computing (TSC). Available at http://tsc.deri.at

[20] J.J. Carroll, Ch. Bizer, P. Hayes and P. Stickler: Named Graphs, Journal of Web Semantics 3(4), 2005