Triple Space Computing Middleware for Semantic Web Services[*]

Omair Shafiq, Reto Krummenacher, Francisco Martin-Recuerda, Ying Ding, Dieter Fensel
Digital Enterprise Research Institute (DERI)
ICT Building, University of Innsbruck,
6020 Innsbruck, Austria.
Email: firstname.lastname@deri.org

Abstract

Triple Space Computing is a communication and coordination paradigm based on the convergence of space-based computing and the Semantic Web. It acts as a global space like middleware to enable communication and coordination based on the principle of publish and read of semantic data. This paper presents overall architecture and its components description to realize the Triple Space Computing. It then presents that how the architecture can act as a middleware to be used by Semantic Web Services as communication paradigm to improve the communication.

1. Introduction

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 these need a solid basement in terms of the underlying semantically enabled communication technologies. Triple Space Computing (TSC) [1] which inherits the publication-based communication model from the tuple space computing paradigm and extends it with semantics provides solutions in such a direction [1]. Instead of sending messages back and forth, TSC based applications will communicate by writing and reading RDF triples in a shared space.

The current Web Service communication model is based on synchronous message exchange. It deviates from the Web principle where the communication model is persistent publishing and reading. Instead of publishing information based on global and persistent URIs, Web Services establish stateful conversations based on the hidden content of messages. The negative side effect of such communication approach is that it requires a strong coupling in terms of access reference, time and location. In this context, Triple Space Computing (TSC), instead of sending messages back and forth among participants as in current message-based technologies, enables applications to communicate by writing and reading RDF triples in a shared persistent information space [1].

The paper is structured as follows: Section 2 provides a description of the Triple Space kernel (TS kernel) architecture (which is a concrete realization of Triple Space Computing concept), components and integration interfaces. Section 3 provides details about the integration of a TS kernel with the Web Service Execution Environment (WSMX) [2] followed by conclusions.

2. TSC Architecture and Components

The TS kernel [4] is the concrete realization of Triple Space Computing concept [1]. It can be used to implement both Triple Space servers and heavy clients and also provides a proxy component, which allows remote access to the kernel. The TS kernel itself consists of multiple components, i.e. operations and security layer, mediation engine, coordination and data access layer.

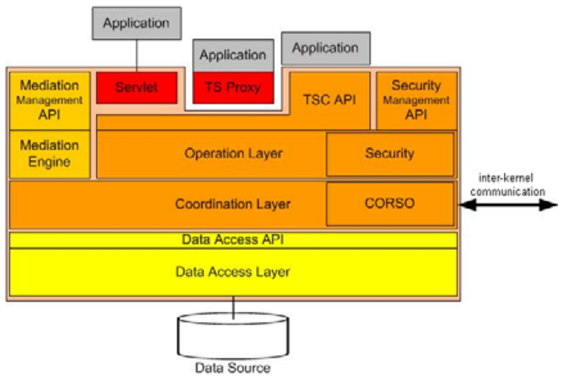

Figure 1: The Triple Space kernel

[*] This work is funded by the Austrian Government under the FIT-IT project TSC (809249).

Figure 1 shows the abstract architecture. The operations and security layer executes Triple Space operations issued by participants. The mediation engine resolves heterogeneity issues by providing mappings for possibly occurring mismatches among different RDF triples. The coordination layer implements transaction management, i.e. the creation, commit and abort of a transaction and guarantees that concurrent operations are processed consistently. The data access layer acts as a gateway to the underlying data storage. The mentioned components are described briefly in subsections below.

Coordination Layer: The responsibility of the coordination layer is to provide multiple TS kernels with a consistent view on the information (in form of named RDF graphs) stored in a Triple Space. The coordination layer is to a big extent based on shared Corso data structures [7] and implemented in form of one Corso runtime per TS kernel. Corso provides transactional access to shared objects, replication strategies to minimize access latency and a notification services. As shown in Fig. 1, the coordination layer provides at once a pipeline between the operation layer and the underlying data access layer and incorporates an existing Corso kernel for TS kernel coordination. In other words the RDF data is forwarded to the data access layer for persistent storage and at the same time replicated and cached in form of Corso objects to serve spaces consisting of more than one TS kernel. A Corso object consists of a set of primitive data fields, references to other objects and is identified by a unique object identifier (OID) that is generated at the time an object is created. TSC defines solely three different objects for the distributed Corso data structures: (1) a SpaceRepository object that aggregates all known Triple Spaces at a TS kernel, (2) a Space object that represents a particular virtual supspace, and (3) a Graph object that models an RDF graph. Corso knows moreover the concept of naming objects and hence the URI that is assigned to every space and every named graph (at the operation layer) is used as object name.

Based on the just introduced Corso structure, the coordination layer has to provide primarily two management functionalities: (1) the discovery of TS kernels (in form of Corso kernels) knowing the same Triple Spaces, and (2) the coordination of data between these TS kernels.

The discovery of related kernels happens via the request for objects by OID. Whenever a kernel has access to one object it can easily get a hand on all related objects. There is basically one interesting approach coming from Corso that works by use of object names; in TSC the URI of a desired space. In order to not reinvent the Internet we aim at using existing DNS facilities to find the IP addresses of interesting kernels that provide access to the desired space. Having such the IP address and hence of course the name of the Space object in form of the URI it is Corso that provides access to all related objects be it other spaces or contained RDF graphs.

Mediation Engine: Several participants 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 support is provided for Triple Space Computing and it is further explained how mismatches occur in RDF data, presents an Abstract Mapping Language [8] 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.

The mediation process in TSC includes specification and processing of mapping rules during the communication. A mapping rule provides link between source and target schema. The mapping rule can also link different particular RDF instances, which results in instance level mediation. 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 process of participants.

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 a participant's desired resources and checks for any possible 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, a 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 the Query Engine and results will be returned to the user.

Data Access and Query Engine: The idea of the data access layer and in particular of the data access interface is to abstract the kernel implementation from the underlying data sources. We aim at an open and flexible framework that allows various different sources of semantic data to be used and integrated into the space infrastructure. The data access API is very similar to the TSC API seen by space users. Instead of manipulating RDF graphs the interface requires also for writing purposes enhanced named graphs [9] and additionally some meta-data about the RDF triples to store. [3]

requires for example publisher information and a timestamp to ensure minimal security measures. Additional information about the graph and the publisher or envisioned consumers, as well as more sophisticated security measures could also be included in the meta-data graph. The write operation looks as follows:

write(URI space, NamedGraph ng, Graph meta) : boolean

There is one meta-graph available per space and its identifier is known by the TS kernel implementation. The meta-data itself is linked to the RDF data graphs by use of the graph name. For the TSC prototype implementation YARS [6] is applied. YARS is an RDF data store that knows the concept of contextualized RDF triple processing. Context is here understood to be the principle of grouping RDF data in particular scopes. In TSC we simply project the graph identifiers onto generated contexts in YARS. These URI – context projections are one of the functionality provided by the prototype data access layer. Moreover the layer provides the actual link to the store by use of the YARS API. The YARS API is heavily inspired by Sun's JDBC API and allows external access to N3 repository manipulation and update commands. Therefore the second major duty of the prototype implementation is the mapping of templates of read operations to N3QL queries. This too is a straightforward procedure, as N3QL relies on the concept of graph patterns that build the TSC templates.

Operations Layer: All Triple Space operations are performed against a certain Triple Space, which is identified by a Triple Space URI. The Triple Space API has been adapted for reading (take, waitToTake, read, waitToRead) and publishing (write) tuples in the Triple Space. A detailed description of this API can be found in [5]. The inability of tuple space computing to provide flow decoupling from the client side [4] is solved by extending the tuples pace computing model with subscription operations. Thus, two main roles for participants are defined: producers, which publish information and advertisements (description of which information will be published); and consumers, which expresses its interest in concrete information by publishing subscriptions. The new extensions based on SIENA API can be found in [5]. Finally, transaction support is included to guarantee the successful execution of a group of operations (or the abortion of all of them if one fails). Transactions have been proposed in several tuple space computing implementations like Triple Space and JavaSpaces. The extensions for transaction support can be found in [5].

3. Triple Space Computing for Semantic Web Services

The currently used communication paradigm in Semantic Web Services (SWS) is synchronous, i.e. users communicate with SWS and SWS communicate with real world Web Services by sending synchronous messages. The problem with synchronous communication is that it requires a quick response as it makes sender halt until the response is received, which is not possible in case of execution process in SWS as it involves heavy processing of semantic descriptions in terms of discovery, selection, composition, mediation, execution. This problem has been overcome by introducing Triple Space Computing as being semantic based asynchronous communication paradigm for communication and coordination of SWS. Web Services Execution Environment (WSMX) is our reference implementation for SWS in which the Triple Space Computing middleware is being integrated. Using Triple Space Computing in WSMX enables to support greater modularization, flexibility and decoupling in communication and coordination and to be highly distributed and easily accessible. Multiple TS kernels coordinate with each other to form virtual space that acts as underline middleware which is used for communication by reading and writing data.

The integration of WSMX and Triple Space Computing is being done in different aspects: (1) enabling components management in WSMX using Triple Space Computing, (2) allowing external communication grounding in WSMX, (3) providing resource management, and (4) enabling communication and coordination between different inter-connected

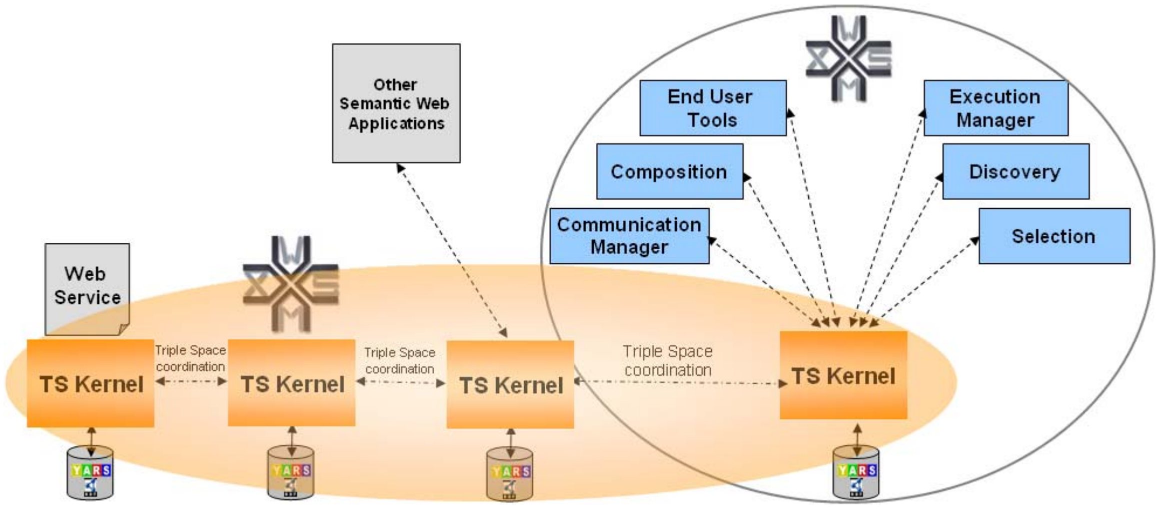

Figure 2: Triple Space Computing Middleware for Semantic Web Services

WSMX systems. Each of the integration aspect is described in the subsections below. In summary, Triple Space Computing acts as a middleware for WSMX, Web Services, different other Semantic Web applications, and users to communicate with each other (Figure 2).

Component management: WSMX has a management component [2] that manages the overall execution by enabling coordination of different components based on some execution scenario specified by user in Goal. In this way there is a clear separation between business and management logic in WSMX. The individual components have clearly defined interfaces and have component implementation well separated from communication issues. Each component in WSMX has a wrapper to handle the communication issues. The WSMX manager and individual components wrappers are needed to be interfaced with Triple Space in order to enable the WSMX manager to manage the components over Triple Space. The communication between manager and wrappers of the components will be carried out by publishing and subscribing the data as a set of RDF triples over Triple Space. The wrappers of components that handle the communication will be interfaced with Triple Space middleware.

External communication grounding: WSMX acts as a semantic middleware between users and real world Web Services. Currently, due to existence of the message oriented communication paradigm, users communicate with WSMX and WSMX communicate with Web Services synchronously. The external communication manager of WSMX is needed to provide a support to communicate over Triple Space. The interfaces for sending and receiving external messages by WSMX are needed provide a grounding support to alternatively communicate over Triple Space. This needs to be resolved by addressing several issues, i.e. invoker component in WSMX is needed to support Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) communication binding over Triple Space. The entry point interfaces will be interfaced with Triple Space middleware in order to provide the glue between existing Web Services standards and Triple Space Computing.

Resource management: WSMX contains different repositories to store Ontologies, Goals, Mediators and Web Services descriptions in the form of WSML files. The internal repositories of WSMX are needed to be made optional and enable to store the WSML based data as set of RDF named graphs in Triple Space Storage. This is mainly concerned with transforming the existing representation of data in form of WSML into RDF triples. The repository interfaces are needed to be interfaced with Triple Space middleware.

Inter-WSMX coordination: After enabling a single WSMX system to rely on Triple Space Computing, the next step is to enable the communication and coordination of different WSMXs over Triple Space, i.e. forming a cluster of different interconnected WSMX systems to support distributed service discovery, selection, composition, mediation, invocation etc. The management component in WSMX has been enhanced to coordinate with WSMX managers in other WSMXs over Triple Space to form a cluster.

4. Conclusion

This paper presents the Triple Space Computing as a middleware for communication and coordination of Semantic Web Services. It describes the overall architecture of Triple Space kernel and describes insight details and the functionality of each of the components. It further explains how the kernel can serve as underline middleware for Semantic Web Services (i.e. WSMX) to improve the communication and coordination of its components, external communication grounding to communicate with service requestors and external Web Services, management of resources in WSMX and finally communication and coordination of multiple inter-connected WSMX systems forming WSMX cluster.

References

[1] D. Fensel: Triple Space computing: Semantic Web Services based on persistent publication of information, the IFIP Int'l Conf. on Intelligence in Communication Systems, 2004.
[2] E. Cimpian and M. Zaremba (eds.): Web Service Execution Environment (WSMX), W3C Member Submission, 2005.
[3] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel: WWW or What Is Wrong with Web Services, Proc. 2005 IEEE European Conf. on Web Services, 2005.
[4] J. Riemer, F. Martin-Recuerda, Y. Ding, M. Murth, B. Sapkota, R. Krummenacher, O. Shafiq, D. Fensel, and E. Kuehn: Triple Space Computing: Adding Semantics to Space-based Computing, 1st Asian Semantic Web Conf., 2006.
[5] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec: The Many Faces of Publish/Subscribe: ACM Computing Survey, 2003.
[6] A. Harth, M. Magni, and S. Decker: Scalable Distributed RDF Storage Infrastructure. DERI Lion Deliverable 1.02, 2005.
[7] E. Kühn: Corso Tutorial, available from http://www.complang.tuwien.ac.at/eva/Download/downloadIndex.html
[8] F. Scharffe: Mapping and Merging Tool Design, DERI OMWG Working Draft, 2005.
[9] J.J. Carroll, Ch. Bizer, P. Hayes, and P. Stickler: Named Graphs, Journal of Web Semantics 3(4), 2005.