Elektrotechnik & Informationstechnik (2007) 000/1/2: 1–8. DOI 10.1007/s00502-006-0408-1

TSC – Triple Space Computing

D. Fensel, R. Krummenacher, O. Shafiq, E. Kühn, J. Riemer, Y. Ding, B. Draxler

Triple Space Computing (TSC) has been proposed as communication and coordination paradigm based on the convergence of space-based computing and the Semantic Web. It acts as a global virtual shared space like middleware to enable communication and coordination of semantic data based on the principle of publish and read. This paper presents an overview of the work in progress under Austrian FIT-IT funded TSC project (*http://tsc.deri.at*). It presents the evolution of the TSC framework, overall architecture and its usage by Semantic Web Services.

Keywords: communication; coordination; middleware; Semantic Web Services; space-based computing

TSC – Triple Space Computing.

Triple Space Computing (TSC) ist ein neuartiges Kommunikations- und Koordinationsparadigma, welches aus einer Kombination von ,,Tuple Spaces'' und Semantic Web-Technologien entstanden ist. Der global zugängliche virtuelle Space stellt eine Middleware zur Verfügung, welche es ermöglicht, semantische Daten via Publizieren und Lesen auszutauschen. In diesem Artikel wird die Entstehung und Entwicklung vom FIT-IT Projekt TSC (http://tsc.deri.at) vorgestellt: die Rahmenbedingungen und Datenmodelle, eine Architektur und Anwendungsbeispiele im Zusammenhang mit Semantic Web Services.

Schlüsselwörter: Kommunikation; Koordination; Middleware; Semantic Web Services; Space-based Computing

Eingegangen am 13. November 2006, angenommen am 23. November 2006

1. Introduction

Aiming at enhancing the facilities for automated information processing on the Internet, Tim Berners-Lee (inventor of the World Wide Web and Director of the W3C) brought up the vision of the Semantic Web. Since existing Web technologies around URI, HTTP, and HTML do not support automated processing of Web content, the aim is to develop technologies that allow describing Web content in a structured manner; furthermore, semantically defined meta-data shall help to overcome the problem of heterogeneity within the Internet as an open and distributed system. Ontologies have been identified as the basic building block for the Semantic Web, as they provide machine-processable, semantic terminology definitions.

In conjunction with the idea of the Semantic Web, Web Services are proposed as the technology for automated information processing, thus combining the benefits of the Web with the strength of component-oriented computation. In fact, Web Services promise to allow automated interaction and seamless integration of several entities of the Web, thus are considered as the technology for next generation information systems with special regard to Enterprise Application Integration, B2B technologies, and e-commerce. As initial Web Service technologies around SOAP, WSDL, and UDDI failed to realize the promise of seamless interoperability, the concept of Semantic Web Services has been conceived. By adding semantics to Web Service descriptions, intelligent inference-based mechanisms shall allow automated discovery, composition, and execution of Web Services.

Space-based computing has its roots in parallel processing. Linda was developed by David Gelernter in the mid-1980s at Yale University. Initially presented as a partial language design (*Gelernter, 1985*), it was then recognized as a novel communication model on its own and is now referred to as a coordination language for parallel and distributed programming. Coordination provides the infrastructure for establishing communication and synchronization between activities and for spawning new activities. There are many instantiations or implementations of the Linda model, embedding Linda in a concrete host language. Examples include C-Linda, Fortran-Linda and Shared-Prolog. Linda allows defining executions of activities or processes orthogonal to the computation language, i.e. Linda does not care about, how processes do the computation, but only how these processes are created. The Linda model is a memory model. The Linda memory is called tuple space and consists of logical tuples. There are two kinds of tuples. Data tuples are passive and contain static data, process tuples or ''live tuples'' are active and represent processes under execution. Processes exchange data by writing and reading data tuples to and from the tuple space.

In 2003 and 2004 there have been discussions and collaborations involving Tim Berners Lee, Dieter Fensel, Eva Kuehn and Frank Leymann on the relationships between the Semantic Web, Web Services and space-based computing. Based on that, Dieter Fensel published a technical report about ''Triple Based Computing'' presenting the idea of a semantically enabled, space-based communication and coordination middleware as an infrastructure for the Semantic Web and Semantic Web Services. These ideas have been adopted for the research project ''Triple Space Computing'' (TSC) funded by the Forschung, Innovation, Technologie – Informationstechnologie (FIT-IT) research programme in the programme line of ''semantic systems and services''. Triple Space Computing inherits the publication-based communication model from the space-based computing paradigm and extends it with

Fensel, Dieter, Univ.-Prof. Dipl.-Ing. Dr., Krummenacher, Reto, Dipl.-Ing., Shafiq, Omair, Bakk. techn., Digital Enterprise Research Institute (DERI), University of Innsbruck, 6020 Innsbruck, Austria (E-mail: dieter.fensel@deri.org); **Kühn, Eva, Ao. Univ.-Prof. Dipl.-Ing. Dr., Riemer, Johannes, Dipl.-Ing.,** Institute of Computer Languages, Department of Computer Science, Vienna University of Technology, 1040 Vienna, Austria; **Ding, Ying, Dr.,** Electronic Web Services (eWS) GmbH, 6020 Innsbruck, Austria; **Draxler, Bernd, Dipl.-Ing.,** Thonhauser Data Engineering (TDE) GmbH, 8700 Leoben, Austria.

semantics. Instead of sending messages back and forth among participants as in current message-based technologies, TSC-enabled applications will communicate by writing and reading RDF triples in the shared space.

2. Triple Space Computing framework

Triple Space Computing (TSC) (*Fensel, 2004*) implies a number of requirements that are not addressed in traditional Linda-like (*Gelernter, 1985*) systems. The requirements are depicted in Table 1. They are mainly concerned with enhancing tuple spaces with Web and Semantic Web technology like resource identification, Semantic Web data models (Resource Description Framework, RDF (*Klyne, Carroll, 2004*)) and query functionality (semantic matching). Moreover, TSC is expected to consider the extended scope of interaction – tuple spaces traditionally serve as communication platform for process and in-house coordination based on a limited number of servers, while TSC aims at a virtually global information space. This results in additional requirements on security, reliability and scalability.

With these requirements in mind, the TSC Framework defines the data models, matching algorithms, interaction APIs and security models at the convergence of space-based computing/shared object spaces and the Semantic Web. The former takes influence on the interaction patterns and data matching, while the latter determined in particular the RDF-based data modeling approach and the storage/query engine installation. In other words TSC borrowed from space-based computing its access primitives, transactional support and eventing/notification mechanism, while the Semantic Web provides the RDF triple syntax and semantics with the resource identification mechanism (URI) and vocabulary separation mechanism (namespaces). Moreover, the RDF query languages heavily influenced the definition of the matching mechanism through SPARQL (*Prud'hommeaux, Seaborne, 2006*) and N3QL (*Berners-Lee, 2004*) technology.

In the continuation of this section a closer look at the interaction API and semantic matching is given. First, however, we explain the core data model concepts.

The interaction API provides all the primitives defined in Linda. The operation names and functionality was, however, mainly influenced by more advanced commercial tuple space products like TSpaces (*Lehmann, McLaughry, Wyckoff, 1999*) and JavaSpaces (*Freeman, Arnold, Hupfer, 1999*). In short the API provides operations for writing, reading, removing in blocking and non-blocking manners. Moreover, convenience methods like update and count were also defined. More detailed descriptions of the different operations are given in Listing 1. Note that, in order to allow Web-like communication, the traditional template-based read and take were enhanced with URI-based primitives that allow the extraction of information by identifier.

In addition to the core API shown in Listing 1 the TSC Framework provides APIs for the publish/subscribe extension, the definition of mediation rules, the management of spaces (creation, destruction), the handling of transaction (commit, rollback) and the definition of roles, permissions and users needed for the security framework. A user is associated with a particular role, while for every role and space the according access permissions can be set.

The semantic template matching mechanisms was motivated by recent achievement in RDF query languages. A template in Linda is a tuple where any of the tuple fields can be replaced by placeholders, so-called variables. In TSC templates are defined to be graph patterns (detailed definition in *(Prud'hommeaux, Seaborne, 2006*)). The graph patterns are RDF in Notation3 (N3, (*Berners-Lee, 2001*)), where variables can take the place of RDF nodes (cf. Listing 2). As graph patterns are at the basis of most RDF query languages (in particular SPARQL) the semantic templates can quite easily be transformed into queries according to the persistence framework (query engine) installed for the Triple Space (cf. Section 3). In that way the semantic templates of TSC provide a simple and extensible

Table 1. Triple Space Computing requirements

Web-like communication	Application and support of established Web technology like URI for resource identification, stateless exchange of information like supported by HTTP
Publishing mechanism	An interaction model based on the publication of information instead on the exchange of messages
Persistent storage	To ensure decoupling in time and in order to provide the 'publish and read'-paradigm the space must ensure persistency of data
Notification mechanism	For improved coordination and process flow decoupling the installation of a notification mechanism (publish-subscribe paradigm) is required
Search and query	Alignment of Linda-like template matching with Semantic Web query languages in order to provide semantic template matching
Trust and security	Any global information space must ensure confidentiality, integrity, non-repudiation and a trust mechanism to ensure a reasonable service middleware
Reliability	The requirement for persistency implies reliable recovery in case of system failure, as well as transactional support for atomic operations
Versioning	Access logging and tracing of changes is very important when sharing dynamic information in public spaces

Table 2. TSC data model concepts

Triple Space	A Triple Space is a uniquely addressable unit of the virtually global information space, i.e., every Triple Space has its own URI. The global space is built by an number of disjoint Triple Spaces (Fig. 1)
Triple	The data model of TSC inherits the syntax (*Klyne, Carroll, 2004*) and semantics (*Hayes, McBride, 2004*) of an RDF triple and the same definitions count
Graph	A graph (RDF graph) is defined in (*Klyne, Carroll, 2004*) as: An *RDF graph* is a set of RDF triples. Here too, TSC inherits the implied semantics
Named Graph	Named graphs (*Carroll et al., 2005*) are the fundamental data unit of TSC and as such all communication is based on named graphs. A named graph is a set of triples named by an URI, i.e. the pair (URI name, RDF graph g)

Listing 1. TSC Interaction API

write (URI ts, Transaction tx, Graph g): URI
The write operation is used to publish an RDF graph to a triple space identified by the URI ts; the graph name is created by the space upon termination of the write operation and the data stored internally as named graph. Transactional write is supported

read (URI ts, Transaction tx, Template t): NamedGraph
take (URI ts, Transaction tx, Template t): NamedGraph
query (URI ts, Transaction tx, Template t): Graph
These three template-based operations are applied to retrieve information from the space. Take has the same semantics as read (retrieval of an entire named graph), however, in a destructive manner. The query primitive on the other hand is used to aggregate all matching triples from the space ts independently of the associated RDF graph, thus it returns a new Graph instead of a whole NamedGraph object. Transactional interaction is supported here, too

read (URI ts, Transaction tx, URI n): NamedGraph
take (URI ts, Transaction tx, URI n): NamedGraph
These two operations have the same semantics as their counterparts introduced above. However, they allow retrieving named graphs by use of their name (URI n)

waitToRead (URI ts, Transaction tx, Template t, TimeOut to): NamedGraph
waitToTake (URI ts, Transaction tx, Template t, TimeOut to): NamedGraph
waitToQuery (URI ts, Transaction tx, Template t, TimeOut to): Graph
The waitTo-operations (the name was taken from TSpaces) provide blocking versions of the retrieval primitives.
While the previously introduced operations return with NULL in case no data was detected, the blocking versions wait until some match is detected or the timeout runs out. The semantics is otherwise precisely as above

update (URI ts, Transaction tx, NamedGraph ng): boolean
Update is on the one hand a convenience method for take and write, and on the other it ensures that graph names are only created by the space. Updates can only be done on graphs that are already known by name to the space. Here too, transactional update is supported

count (URI ts, Transaction tx, Template t): long
Count provides the exact same functionality as a loop with counter over a query operation. Note that count only provides an estimate; just as all other primitives the returned set of triples is not ensure to be complete

Listing 2. Graph pattern-based semantic template

?s a doap:Project; foaf:member ?o	This graph pattern queries all triples where the subject is of type doap:Project and where the same subject has triples indicating the members
?s ?p ?o. ?o a foaf:Person	This template matches all triples where the object is of type foaf:Person
?s foaf:name ?a; foaf:mbox ?b	This last template matches the triples that contain subjects for which the name and a mailbox (foaf:mbox) are indicated

means to match data in a Triple Space, analogue to tuple templates in Linda.

3. Triple Space kernel

Like with the Web, the TSC project proposal aimed at building a Triple Space Computing infrastructure based on the abstract model called REST (Representational State Transfer) (*Fielding, 2000*). The fundamental principle of REST is that resources are identified by URIs and accessed via a stateless protocol like HTTP in order to transfer representations, such as HTML or XML documents, of resources over the network. HTTP provides a minimal set of operations enough to model any applications domain (*Fielding, 2000*).

Since every representation transfer must be initiated by the client, and every response must be generated as soon as possible (the statelessness requirement) there is no way for a server to transmit any information to a client asynchronously in REST. Furthermore, there is no direct way to model a peer-to-peer relationship (*Khare, Taylor, 2004*) between clients. Finally, HTTP caching based on expiration times for cached requests is not applicable in TSC; where a server has no pre-knowledge of the lifetimes of named graphs. The limitations of REST in the context of TSC motivated our approach of a hybrid architecture called super-peer architecture, which combines traditional client/server and peer-to-peer architectures. In this architecture there are three kinds of nodes: servers, heavy clients and light clients. In the simplest configuration, a particular Triple Space is realized by a single server, which is accessed by multiple light clients, for example via HTTP, in order to write and read named graphs and to receive notifications about graphs of interest. As the number of light clients increases, the server may become a bottleneck. To overcome this, additional servers can be deployed to provide additional access points to a Triple Space for light clients. As a result, a single Triple Space is be effectively spanned by multiple servers, which use an inter-server protocol to consistently distribute and collect named graphs to and from other involved servers. Servers can also be deployed to act as caching proxies in order to improve clients-perceived access times. The third kind of nodes is heavy clients, which are not

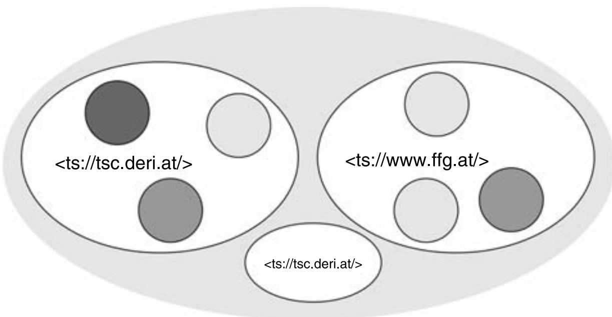

Fig. 1. Definition of the Triple Space

always connected to the system. Like servers they are capable to store and replicate Triple Spaces and support users and applications to work off-line with their own replicas. While heavy clients can join existing Triple Spaces spanned by servers, they are not forced to do so.

The core functionality of TSC servers and heavy clients is realized by a component called Triple Space kernel (TS kernel). 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 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 like HTTP, as already mentioned above. In this case a server side component, e.g. a servlet, translates the protocol to the native TS kernel interface. Figure 2 shows the architecture of the TS kernel. Main components of the TS kernel will be briefly described in subsections below.

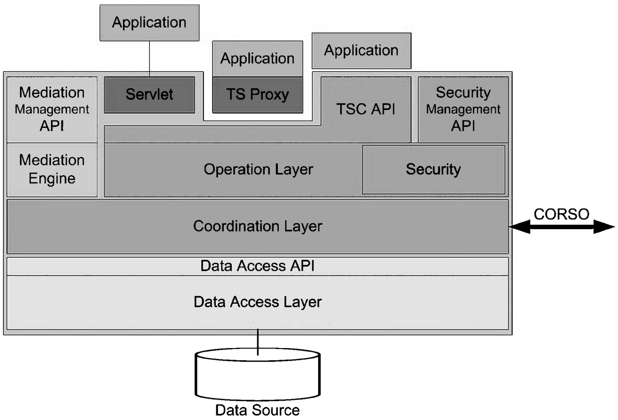

Fig. 2. Triple Space kernel architecture

3.1 Mediation engine

Due to diversity in the nature of different communicating participants over Triple Space, the possibility of the heterogeneity in the data used for communication of different participants may arise and make mediation an important issue to be resolved in the Triple Space Computing. The Mediation Engine (*Shafiq et al., 2006b*) as part of the TS kernel (*Riemer et al., 2007*) is concerned with handling this heterogeneity by resolving possibly occurring mismatches among different triples. Assume two TSC participants using different data models for communication. Then an RDF instance in an 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 transform the RDF triples according to different RDF Schemas of different communicating participants. The mediation rules are to be specified at design time and will be processed by a mediation engine at runtime.

The TSC mediation engine starts working when users add mediation mapping rules via mediation management interface. Rules are defined in the Abstract Mapping Language (AML) (*Scharffe, de Bruijn, 2005*) which is independent of any programming language and is able to model complex correspondences that may stand between two ontologies. Graphical user interfaces are available to define rules in AML. In TSC mediation rules are themselves stored in Triple Spaces as RDF graphs. As a result, rules created at one server or heavy client can be shared with all other nodes spanning a Triple Space. To represent mediation rules in RDF, an RDF grounding for AML was defined. A component called mediation manager implements serialization of rules and helps adding, replacing and deleting mapping rules.

3.2 Coordination layer

The coordination layer has three responsibilities, (1) local TS operations, such as reading and writing named graphs are executed by accessing the local data access layer and by consistently propagating changes to other involved TS kernels, (2) changes of a space originating from other TS kernels are recognized and applied to the local data access layer, and (3) remote TS kernels involved to span a certain space are discovered automatically in the network.

Consistent concurrent access to named graphs is provided via transactions. In principle both optimistic and pessimistic transactions are applicable for TSC; however, they are not exchangeable due to differences in their semantics. We decided to support optimistic transactions, because they provide a higher degree of concurrency, if read operations are more frequent than write operations, which results in a higher throughput, because they are free of deadlocks without the introduction of additional, semantically sophisticated timeout parameters and finally, because they enable a pragmatic integration of a data access layer, which itself does not support a transaction interface.

The prototype implementation of the coordination layer is based on the CORSO (Coordinated Shared Objects Spaces) (*Kühn, 1994*) middleware. CORSO is a peer-to-peer implementation of a virtual shared data space, which allows reading and writing structured, shared data objects. It has a built-in distributed transaction manager and distributes spaces via an asynchronous, primary-based replication protocol. In the TSC prototype, Triple Spaces and named graphs are mapped to distributed CORSO data structures. TSC operations like reading and writing named graphs are translated to algorithms on these CORSO data structures. CORSO further provides a notification mechanism to get informed about changes in the shared space. The coordination layer uses CORSO notifications to react on inserted or removed named graphs and to asynchronously update the underlying data access layer. The discovery of TS kernels involved in spanning a Triple Space is based on the Domain Name System (DNS) for wide area networks and on a new protocol based on UDP-multicast and CORSO for local area networks.

3.3 Data access layer

Any Triple Space implementation requires a storage and retrieval framework to (1) ensure the desired persistency, (2) to support semantic template matching based on Semantic Web query languages and (3) to provide at least a limited amount of reasoning. In order to bind arbitrary data stores and query engines to TS kernels we define a Data Access Layer (DAL) which defines operations for storing, retrieving and deleting RDF graphs.

The prototype implementation of the data access layer is based on YARS (Yet Another RDF Store) (*Harth, Decker, 2005*), a lightweight persistence framework developed in Java at DERI Galway which uses optimized indexes for better query performance. Besides the noteworthy performance, the fact that the consortium has access to the source code and the implementers through DERI Innsbruck, YARS has in particular be chosen as it is constructed to store quads or contextualized triples instead of plain RDF triples. This allows for direct usage of the chosen data model based on named graphs (*Carroll et al., 2005*).

One of the main tasks of the data access layer is to translate templates into N3QL queries for YARS. To keep the Data Access API (DAPI) as simple as possible it only defines one operations to retrieve data: retrieve(URI ts, Transaction tx, Template t):Graph. The TSC API, however, allows a space user to retrieve data either based on templates or by use of the graph name. As the DAPI does not directly support an interface for URI-based retrieval it is also necessary to adapt the operation layer in order to transform those requests into templates. First, the URI has to be packed into a graph pattern template according to (*Riemer et al., 2006*) as part of the Operation Layer processing. The request is then forwarded in form of the template to the DAL, where the template is transformed into a N3QL query that can be sent to the YARS servlet.

4. Triple Space Computing for Semantic Web Services

Semantic Web Services have been emerged to enable dynamic Web Service discovery, composition and execution by using semantic descriptions using ontologies as its basis. The Semantic Web Services framework has been envisioned (*Fensel, Bussler, 2002*). It provides an ontology, called Web Service Modeling Ontology (WSMO) (*Roman et al., 2006*), a language, called Web Service Modeling Language (WSML) (*Roman et al., 2006*), which provides a formal syntax and semantics for WSMO, and an execution environment, called Web Service Execution Environment (WSMX) (*Roman et al., 2006*), which is a reference implementation for WSMO, offering support for interacting with SWS.

The currently used communication paradigm in Semantic Web Services (SWS) (*Fensel, Bussler, 2002*) 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. The 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 a 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 has been proposed in four major aspects (*Shafiq et al., 2006a*): (1) enabling component management in WSMX using Triple Space Computing, (2) allowing external communication grounding in

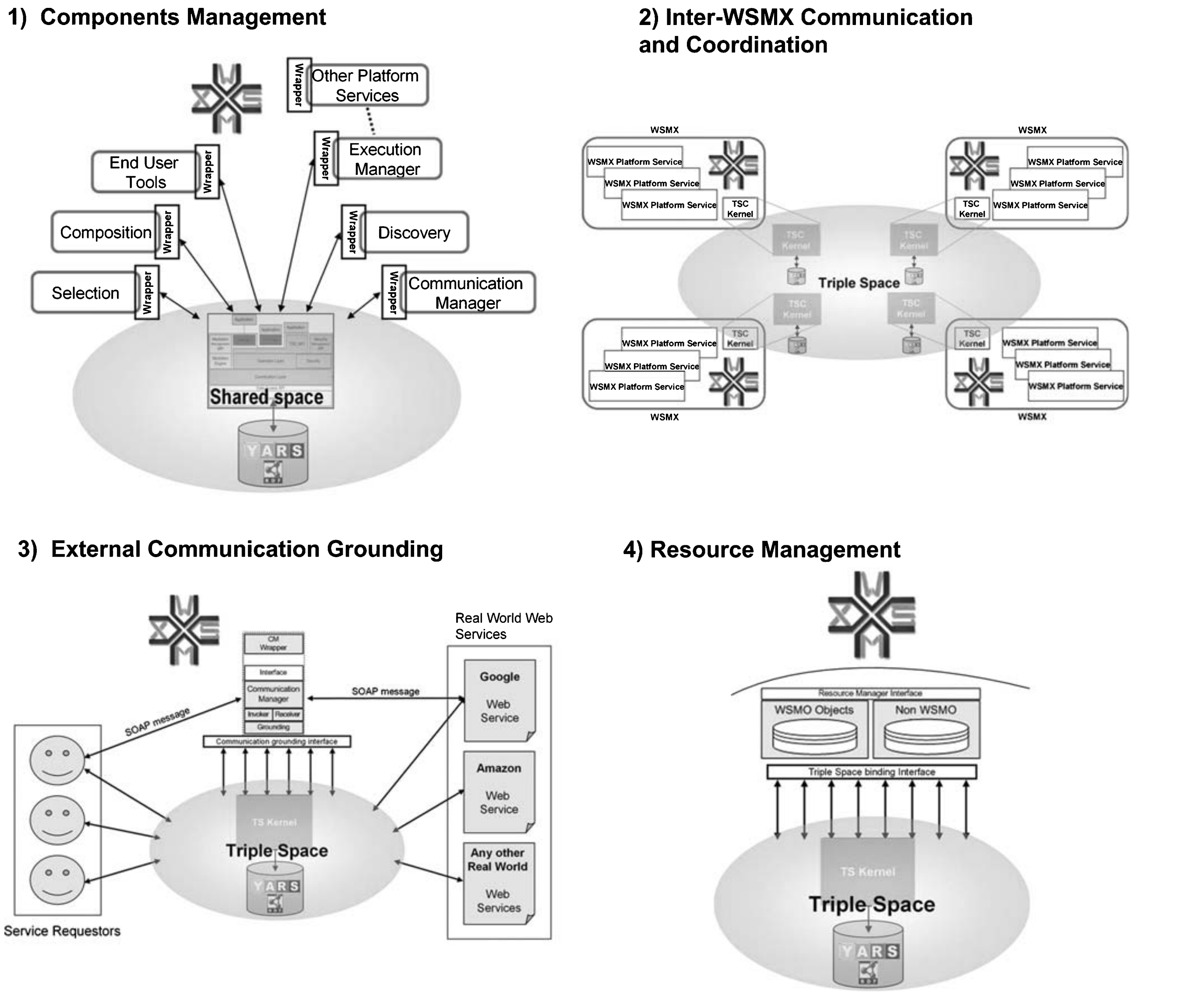

Fig. 3. Triple Space Computing for WSMX

WSMX, (3) providing resource management, and (4) enabling communication and coordination between different inter-connected 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 3 shows an initial architecture of each integration aspect.

4.1 Component management in WSMX using Triple Space Computing

WSMX has a management component that manages the over all execution of the system by coordinating different components based on dynamic execution semantics. In this way there has been made a clear separation between business and management logic in WSMX. The individual components have clearly defined interfaces and have component implementation well separated with 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 graphs over triple space. The wrappers of components that handle communication will be interfaced with Triple Space middleware. The WSMX manager has been designed in such a way that it could distinguish between the data flows related with the business logic (execution of components based on the requirements of a concrete operational semantic) and the data flows related with the management logic (monitoring the components, load-balancing, instantiation of threads, etc.).

There are two ways for WSMX components to access a TS core, i.e. heavy clients embed the TS core as a Java package and the application and TS core run in the same Java Virtual Machine. In this case CORSO (*Kühn, 1994*) and YARS (*Harth, Decker, 2005*) runtimes need to be deployed together with the heavy client application. The second way is to deploy a standalone TS kernel as a server, which may be accessed by multiple light clients via remoting. Both scenarios can work. However, we recommend using light clients in case of communication and coordination within the WSMX system as in such case it will make the keep the complexity level of components wrapper and the access of light client embedded in wrappers will be local to the Triple Space kernel.

4.2 Multiple WSMX instances interconnection using Triple Space Computing

After enabling WSMX Manager to perform communication and coordination of components internally, the next step will be to enable the communication and coordination of different WSMXs over Triple Space, i.e. forming a cluster of different interconnected WSMX nodes to support distributed service discovery, selection, composition, mediation, invocation, etc. The communication model used in the current implementation of WSMX is synchronous. Synchronous communication is beneficial when immediate responses are required. Since WSMX is dealing with Web service Discovery, Mediation and Invocation, immediate responses are usually not available. In such situations, the synchronous communication will be costly as it forces the system (component) to remain idle until the response is available. In order to minimize such overhead imposed by synchronicity, Triple Space can serve as a communication channel between WSMXs thereby introducing synchronicity between communicating parties. The Triple Space supports purely asynchronous communication that optimizes performance as well as communication robustness.

The figure above shows the idea of having different WSMX systems to be interconnected to each other over Triple Space. This will help the WSMX in providing distributed service discovery, selection, composition, mediation and invocation. There can be the possibility that different WSMX systems are running at different location over the globe containing different information (i.e. semantic description of commercial Web Services, mediation rules, ontologies and goals). The service requestor local to a particular WSMX will not be aware of other WSMX systems and the data contained by other WSMX systems. In this case, it will enable different WSMX systems to be aware of each other and to access the data of other WSMXs over Triple Space, or redirect the goals to other WSMXs.

4.3 External communication grounding in WSMX using Triple Space Computing

WSMX acts as a semantic middleware between users and real world Web Services. Currently, due to existence of 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 to 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.

The Communication Manager will be provided with Triple Space based grounding support. It will help in providing an additional or alternative Triple Space based access interface to access WSMX. It will enable Triple Space clients to submit Goals to WSMX via Triple Space which will bring the real sense of asynchronous communication of Triple Space because normally Goal execution in WSMX (performing service discovery, selection, composition, mediation and invocation) takes significant amount of time. When the service requestors will be able to submit the Goals to WSMX over Triple Space, it will not make them hang-up with WSMX until the Goal has been executed and will make the communication process of users with WSMX more flexible and reliable.

4.4 Resource management in WSMX using Triple Space Computing

WSMX contains different repositories to store ontologies, goals, mediators and Web Services descriptions as WSML based 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 representation. The repository interfaces are needed to be interfaced with Triple Space middleware. The Resource Manager in WSMX currently manages the persistent storage of data in the repositories. The Resource Manager provides a heterogeneous interface for WSMX. The component implementing this interface is responsible for storing every data item WSMX uses. The WSMO API provides a set of Java interfaces that can be used to represent the domain model defined by WSMO. WSMO4J (*http://wsmo4j.sourceforge.net*) provides both the API itself and a reference implementation. Currently WSMX defines interfaces for six repositories. Four of these repositories correspond to the top level concept of WSMO, i.e. Web Services, ontologies, goals, and mediators. The fifth repository is for non-WSMO data items e.g. events and messages. Finally, the sixth repository stores WSDL documents used to ground WSMO service descriptions to SOAP.

The storage of WSMO top level entities on Triple Space will help in enhancing and fastening the process access of the data items afterwards. For instance, in the current discovery mechanism of WSMX, the WSML reasoners have to reason on each and every Web Service description available in the local repositories which takes significant amount of time. When the Web Services descriptions will be stored over Triple Space, the template matching based simpler reasoning will be used as a first step in order to filter-out the most relevant and possibly required Web Service descriptions. The filtered Web Services descriptions based on template based matching over Triple Space are retrieved and converted back to WSML to be used by WSML reasoners. It makes the process of discovery simpler and faster by performing reasoning operations only on relevant Web Service descriptions rather than all.

5. Conclusions

In this paper we provide an overview of overall work in-progress in Triple Space Computing (TSC) project funded by the Austrian Government under the program FIT-IT Semantic System. In the project we are building the Triple Space Computing as a communication and coordination framework for semantic technologies. In this paper we presented the background of TSC, introduction, state-of-the-art, TSC framework, data and interaction model, TSC architecture and TSC integration with Semantic Web Services. The project has entered into its final phase where theoretical work has been completed and currently prototypes are under development which will be followed by extensive evaluation.

Acknowledgements

The work is funded by the Austrian Government under the program FIT-IT (Forschung, Innovation, Technologie – Informationstechnologie), project 809249 TSC – Triple Space Computing.

References

Berners-Lee, T. (2001): Notation 3: An RDF language for the Semantic Web. W3C Design Issues, November 2001, http://www.w3.org/DesignIssues/Notation3.html.

Berners-Lee, T. (2004): N3QL – RDF Data Query Language. W3C Design Issues, July 2004, http://www.w3.org/DesignIssues/N3QL.html.

Bussler, C. (2005): A minimal triple space computing architecture. Proc. of the 2nd WSMO Implementation Workshop (WIW 2005), Innsbruck, Austria, June 2005.

Carroll, J. J., Bizer, Ch., Hayes, P., Stickler, P. (2005): Named graphs. Journal of Web Semantics 3 (4).

Fensel, D. (2004): Triple-space computing: Semantic web services based on persistent publication of information. Proc. of the IFIP Int. Conf. on Intelligence in Communication Systems, INTELLCOMM 2004, Bangkok, Thailand, November 2004.

Fensel, D., Bussler, C. (2002): The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1 (2).

Fielding, R. T. (2000): Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine.

Freeman, E., Arnold, K., Hupfer, S. (1999): JavaSpaces: principles, patterns, and practice. The Jini Technology Series, Addison-Wesley Longman.

Gelernter, D. (1985): Generative communication in Linda. ACM Transactions in Prog. Languages and Systems (TOPLAS) 7(1).

Harth, A., Decker, S. (2005): Optimized index structures for querying RDF from the Web. Proc. of the 3rd Latin American Web Congress, Buenos Aires, Argentina, October/November 2005.

Hayes, P., McBride, B. (eds.) (2004): RDF Semantics. W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-mt/.

Khare, R., Taylor, R. N. (2004): Extending the Representational State Transfer (REST) Architectural style for decentralized systems. Proc. of the Int. Conf. on Software Engineering (ICSE), Edinburgh, Scotland, May 2004.

Klyne, G., Carroll, J. J. (eds.) (2004): Resource Description Framework (RDF): concepts and abstract syntax. W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-concepts/.

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

Kühn, E. (1994): Fault-tolerance for communicating multidatabase transactions. Proc. of the 27th Hawaii Int. Conf. on System Sciences (HICSS), Wailea, Maui, Hawaii, January 1994.

Kühn, E., Beinhart, [redacted]., Murth, M. (2005): Improving data quality of mobile Internet applications with an extensible virtual shared memory approach. Proc. of Iadis www/Internet 2005 Conference, Lisbon, December 2005.

Lehman, T. J., McLaughry, St. W., Wyckoff, P. (1999): TSpaces: The Next Wave. Proc. of the Hawaii Int. Conf. on System Sciences (HICSS-32), Wailea, Maui, Hawaii, January 1999.

Prud'hommeaux, E., Seaborne, A. (eds.) (2006): SPARQL Query Language for RDF. W3C Candidate Recommendation, April 2006, http://www.w3.org/TR/rdf-sparql-query/.

Riemer, J., Martin-Recuerda, F., Ding, Y., Murth, M., Sapkota, B., Krummenacher, R., Shafiq, O., Fensel. D., Kuehn, E. (2006): Triple Space Computing: adding semantics to space-based computing. Proc. 1st Asian Semantic Web Conf., Beijing, China, September 2006.

Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Bussler, C., Fensel, D. (2006): WWW – WSMO, WSML, and WSMX in a nutshell. Proc. of the 1st Asian Semantic Web Conf. (ASWC 2006), Beijing, China, September 2006.

Sapkota, B., Kilgarriff, E., Bussler, C. (2006): Role of Triple Space Computing in Semantic Web Services. Proc. of the 8th Asia Pacific Web Conf. (APWEB 2006), Harbin, China, January 2006.

Scharffe, F., de Bruijn, J. (2005): A language to specify mappings between ontologies. Proc. of 1st Int. Conf. on signal-image technology and Internet-based systems (SITIS 2005), Yaounde, Cameroon, November 2005.

Shafiq, O., Krummenacher, R., Martin-Recuerda, F., Ding, Y., Fensel, D. (2006a): Triple Space Computing Middleware for Semantic Web Services. Proc. of 2006 Middleware for Web Services (MWS 2006) Workshop at the 10th Int. IEEE Enterprise Computing Conf. (EDOC 2006) in Hong Kong, October 2006.

Shafiq, O., Scharffe, F., Krummenacher, R., Ding, Y., Fensel, D. (2006b): Data mediation support for Triple Space Computing. Proc of the 2nd IEEE Int. Conf. on Collaborative Computing (CollaborateCom 2006), Atlanta, GA, USA, November 2006.

WSMO4J: an API and a reference implementation for building Semantic Web Services applications compliant with the Web Service Modeling Ontology (WSMO), http://wsmo4j.sourceforge.net/.

The Authors

Dieter Fensel
is Director of Digital Enterprise Research Institute (DERI) at the Leopold Franzens University of Innsbruck, Austria. Prior to this he was Director of Digital Enterprise Research Institute (DERI) at National University Ireland at Galway, Ireland. Moreover, he has held positions at University of Karlsruhe (AIFB), University of Amsterdam (UvA), and the Vrije Universiteit Amsterdam (VU). His current research interests are around the usage of semantics in 21st century computer science. He received his PhD from University of Karlsruhe in 1994.

Reto Krummenacher
holds a degree in communication system engineering from the Swiss Federal Institute of Technology in Lausanne, Switzerland (EPFL). He concluded his master studies in 2004 with a thesis on prefetching for location-aware mobile information systems which was conducted at Fraunhofer's Integrated Publication and Information Systems Institute IPSI in Darmstadt, Germany. Since late 2004 he is, as PhD student, part of the Ubiquitous Services Working Group at the Digital Enterprise Research Institute of the University of Innsbruck (DERI Innsbruck) with a strong focus on Triple Space Computing in ubiquitous computing environments.

Omair Shafiq
is working as junior researcher at the Digital Enterprise Research Institute (DERI), University of Innsbruck, Austria. Prior to this he was working as research associate in the Semantic Grid and Multi Agent Systems Research Group in the Information Technology department of National University of Sciences and Technology (NUST), Rawalpindi/Islamabad, Pakistan. He received his integrated degree of BS in Computing with honors and Rector's Gold Medal award in June 2005 from the same university. He was also serving as technical leader of joint research initiative between NUST Pakistan, Communication Technologies (Comtec), Sendai, Japan and University of Arkansas at Little Rock (UALR), AR, USA till August 2005. His current research interests focus on Semantic Web, Semantic Web Services, Semantic Grid, Peer to Peer Computing and Distributed Systems Management.

Eva Kühn
holds the titles graduated engineer of computer sciences (Dipl.-Ing.), Ph.D. (Dr. techn.) and Venia Docendi (Univ.-Doz.) from the Vienna University of Technology, Vienna, Austria. She received the Heinz-Zemanek research award for the her Ph.D. work on ''Multi Database Systems'' and a Kurt-Gödel Research Grant from the Austrian Government for a sabbatical at the Indiana Center for Databases at Purdue University, Indiana, USA. Dr. Kühn is employed as A.o. Univ.-Prof. at the Vienna University of Technology, Institute of Computer Languages. Her current teaching topics are virtual shared memory systems and coordination tools. Since 2006 Dr. Kühn is a member of the Senate of the Christian Doppler Forschungsgesellschaft.

Johannes Riemer
studied computer science at the Vienna University of Technology and received a degree in Computer Science (Diploma) in 2003. During his studies his special interests have been on distributed systems, real-time systems and object oriented languages. Parallel to university he has been working in industry since 1999, where he gained experience in design and development of distributed applications and virtual shared memory middleware. Since 2005 he is a PhD student at the Vienna University of Technology with a research focus on space-based computing, virtual shared memory, triple space computing, semantic web and service oriented architecture.

Ying Ding
is senior consultant in Electronic Web Services GmbH. Prior to this she was a researcher at the Division of Mathematics and Computer Science at the Free University of Amsterdam. She completed her Ph.D. in School of Applied Science, Nanyang Technological University, Singapore. She has been involved in various European-Union funded projects and served as consultant in many projects between universities and the companies. Her current interest areas include semantic web, web services, semantic information retrieval, knowledge and content management, e-commerce and commercial application of semantic web technology.

Bernhard Draxler
is working as software developer and designer for data acquisition software based on java/C++ for windows and Embedded Linux with SQL-databases at Thonhauser Data Engineering GmbH, Leoben, Austria. He received his diploma in 1997 on Vienna University of Technology. Prior to this current position, he worked on virtual shared memory system for windows and Unix Linux at Tecco software development CORP, at IBM Austria on development of workflow management systems and at Vienna software publishing team leader development of an Office package for OS/2.