A Minimal Triple Space Computing Architecture

Christoph Bussler

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway, Ireland

Chris.Bussler@DERI.org

Abstract. The visionary approach of Triple Space Computing was recently introduced based on the insight that Web Services do not follow the Web paradigm of 'persistently publish and read' [Fensel, 2004]. Instead, Web Services currently require a synchronous connection to transmit data transparently bypassing and ignoring the power of the Web paradigm. Triple Space Computing proposes to publish communication data analogous to the publication of Web pages: persistently for anybody to read who has access to it at any point in time. This has several benefits. The provider of data can publish it at any point in time (time autonomy), independent of its internal storage (location autonomy), independent of the knowledge about potential readers (reference autonomy) and independent of its internal data schema (schema autonomy). This article introduces a minimal Internet-scalable Triple Space Computing architecture based on Semantic Web technology that implements these four types of autonomy in the simplest way possible with as minimal functionality as feasible to be useful with no or almost no impact to publishers and reader of communication data.

1 Introduction to Triple Space Computing

The visionary approach of Triple Space Computing (TSC) was recently introduced based on the insight that Web Services do not follow the Web paradigm of 'persistently publish and read' [Fensel, 2004]. Anybody can define a Web page persistently and publish it. Both, the publication and the reading can be done independently of each other, i.e., asynchronously. Fensel proposes to follow exactly this paradigm for the communication of data between software systems across the Internet: publish the data persistently and make it available for reading it[1]. The location for storing and accessing data is called triple space, and, as it will be later introduced, it is a virtual storage location.

[1] Note on terminology: sometimes authors use the term 'publish' for providing communication data and 'consume' for accessing it. The term 'consume', however, has the connotation of destroying the read data. However, like in the Web pages case, a read does not destroy as such; it only creates another copy for the reader.

Instead of following the 'persistently publish and read' paradigm, traditional Web Services based on the Web Service Definition Language (WSDL[2], [Christensen et al., 2001]) and the Simple Access Object Protocol (SAOP[3], [Mitra, 2003]) require a synchronous Hypertext-Transfer-Protocol (HTTP[4], [Fielding et al., 1999]) connection to transmit data transparently bypassing the power of the Web paradigm. This means that traditional Web Services require the sender and receiver of data to have a tight same-time synchronous connection, to agree on the data format, to know each other and share a common representation. Besides violating the Web paradigm of 'persistently publish and read', this also prevents general mechanisms like Web caching and unique referencing of data through Uniform Resource Identifiers (URI[5], [Berners-Lee et al., 2005]).

Triple Space Computing establishes the mechanism to publish communication data according to the Web paradigm of 'persistently publish and read' and in this way TSC brings machine-to-machine Web Service communication to the Web in its real sense: 'Web' Services. This has several benefits. The provider of data can publish it at any point in time (time autonomy), independent of it internal storage (location autonomy), independent of the knowledge about potential readers (reference autonomy) and independent of it internal data schema (schema autonomy):

- **Time autonomy**. Time autonomy means that there are no time dependencies between the data provider and reader. Each can at its own discretion access the triple space and write or read data.
- **Location autonomy**. The triple space as a storage location is independent from any storage space in the provider or reader of data. Complete independence is achieved by ensuring that triples are passed to and from the triple space by value and in the format required by the triple space.
- **Reference autonomy**. Provider and reader of data might know about each other, but do not have to know each other for purposes of communication through the triple space. In the simplest case, the reading and writing of data is anonymous. Therefore, TSC does not enforce reference but allows reference autonomy.
- **Data schema autonomy**. TSC provides its own data schema (it is going to be based on triples according to RDF [Manola et al., 2004]) and the data written and retrieved from a triple space will follow the TSC data model. This makes the provider and reader independent of their internal data schema they have and that might be very different.

TSC are based on a combination of proven technology. As it will be discussed later in more detail in this article, TSC will be based on the HTTP protocol, the Resource Description Framework (RDF[6], [Manola et al., 2004]) technology and commercially available storage components like databases or file systems. In that regard, exactly the same technology is used as in case of (Semantic) Web pages.

---

[2] http://www.w3.org/TR/wsdl
[3] http://www.w3.org/TR/soap/
[4] http://www.ietf.org/rfc/rfc2616.txt
[5] http://www.ietf.org/rfc/rfc3986.txt
[6] http://www.w3.org/RDF/

The goal of the paper is to define a minimal Internet-scalable Triple Space Computing architecture based on Semantic Web technology that implements the four types of autonomy (time, location, reference, data schema) in the simplest way possible to be useful with no or minimal impact to publishers and consumers of communication data. The emphasis is on 'minimal' and 'simple' for purpose in order to establish the minimal functionality required implementing the four types of autonomy.

Section 2 introduces the architecture concepts for TSC. Section 3 details the various architecture components necessary as well as their interactions from a dynamic perspective. Section 4 lists many implementation suggestions in order to utilize existing technology and knowledge to the extent possible. Section 5 discusses briefly related work and Section 6 summarizes.

2 Architecture Concepts

The architecture concepts are introduced first before the components of the architecture are discussed. This allows the discussion of the major elements without having a particular modularization and implementation in mind.

2.1 Storage Object

The object that is written to a triple space and read from it (possible numerous times) is a RDF triple as defined in [Manola et al., 2004]. A triple has three parts to it, a subject, an object and a predicate. Triples are uniquely identified through URIs [Berners-Lee et al., 2005]. This means that each triple in any triple space is uniquely marked and can be distinguished from all the other triples by its unique URI.

Between the URI of a triple and the URI of the triple space it is located in is no direct relationship in form of any URI encoding. That is, the URI of the triple space is not encoded in the URI identifying the triple itself. This means that it is not possible to determine triples in a triple space by the URI construction (or 'guessing'). In order to determine all the triples that are located in the triple space, a reader has to retrieve them all. In order to do this, however, the reader has to already know all triples. The reason for this is that this is the minimal functionality.

Triples can refer to each other in order to express more complex data structures in order to describe more complex information. In its simplest form, the triple space does not recognize this fact. All triples are individual objects and any relationship between those is ignored. For a reader, however, this is not a problem. Once a reader reads a triple, the reader then can interpret the contents of the triple. If it recognizes that a triple refers to another one, the reader can then read this one, too. This way the graph of triples can be reconstructed by the reader and the minimal architecture does not get into the way of achieving a perfect recall.

2.2 Storage Location and Location Virtualization

A triple space is a virtual space. A triple space is identified through a unique URL. Triples are written and read with this URL as triple storage location. A triple space as such is not associated in a particular way with an implementation called triple space server. An implementation, of course, has to provide a physical storage location like a database or file system directory or directory itself. However, one implementation can 'host' many triple spaces. A triple space has to be part of one implementation, but one implementation can host many triple spaces. The relationship is one-to-many between a triple space server and (virtual) triple spaces.

For example, **`www.deri.org/incoming_resumees`** might point to a triple space that prospective applications use to submit their résumés. **`www.deri.org/phone_numbers`** might point to a triple space that contains all phone numbers of all members of DERI. Both are virtual triple spaces in the sense that they are storing and managing 'their' triples. From an implementation and deployment viewpoint both might be hosted by the same implementation (e.g. if DERI would only have one triple space server). However, it might also be that both are hosted by different triple space servers. From a user perspective this is completely transparent as triple spaces are virtual entities and no reference to a specific server is necessary when invoking operations on a triple space to write or to read triples.

2.3 Storage Operations

Triples are written into a triple space by value. All three elements of a triple have to be specified and the concept of object references (data passing by reference) does not apply. Hence the write operation (in informal syntax) is very straight forward:

– `write (triple)`

The precise syntax is defined in the triple space transfer protocol, see Section 0.

Reading a triple is equally simple. A triple is read from a given triple space identified by a URL by invoking the read operation with the identifying triple URI as parameter. Hence the read operation in informal syntax is:

– `read (URI)` **`returns`** `triple`

The read operation is non-destroying, i.e., a copy of the identified triple is returned and the triple still remains in the triple space (analogous to Web pages that are not destroyed through the first reader). Also, the returned triple is not a replica, meaning, that if the triple in the triple space changes, any retrieved copy of it is unaffected by that modification.

2.4 Storage Semantics

After a triple space is installed it is empty. No triple is contained in the triple space.

- **Write**. The write operation on a triple space can encounter two states of the triple space. In one state no triple exists with the same URI. In this case the triple is added to the triple space. In the other state a triple with the identical URI already exists. In this case the existing triple will be overwritten with the one identified in the write operation. The original values will be lost. This semantics ensures that the write operation always succeeds.
- **Read**. The read operation can encounter also two states. In one state a triple for the given URI exists and in this case a copy of the triple will be returned. The read is non-destroying and the triple can be read again any number of times. In the second case no triple exist for the given URI. In this case an error code is returned indicating that no triple exists in the triple space with the given URI. The error code is distinct from an empty triple where all three values are empty. So the correct definition of the read operation is:
  - `read (URI) returns triple | error_code`
- **Concurrent Writes**. A triple space can have any number of triple writers. They can try to write different triples or even the same triples (i.e., triples with the same URI). In any case, the triple space has to ensure that no lost updates occur, meaning, that writers are serialized. Only one triple writer can write a triple at the same time. If appropriate underlying technologies are used, this semantics is very easy to implement, e.g., through transactions. Since only URIs are the distinguishing features of triples from a TSC viewpoint, triples with different URIs can be written concurrently, if the underlying technology allows it. However, this is an implementation detail, not a conceptual concern.
- **Concurrent Reads**. Since the reading of triples is non-destroying, concurrent reading of triples with the same URI or different URIs is possible. Any number of readers can read triples concurrently at any point in time. Of course, this is only possible if the underlying technology allows this implementation. If the underlying technology serializes read operations, then concurrent reads only exist conceptually, but not in fact.
- **Concurrent Write and Read Operation Interleaving**. Finally, triple spaces can be access by writers and readers concurrently, too, at least they can attempt it. In this case it must be ensured that readers cannot retrieve ‘half written’ triples, i.e., the granularity of the storage operations must be the reading or writing of a whole triple. If the granularity is finer then a complete triple, then it might be that a reader can read the intermediate state of a write or an update (the case where a triple is replaced by one with new values). In this case the triple server must ensure that triple storage operations are serialized according to the reader-writer problem.

2.5 Summary

In summary, the conceptual model of the minimal TSC architecture is small and concise. This promises an efficient and simple architecture as well as an outstanding performing implementation. Both are show in the next two sections.

3 Architecture Components and Component Interaction

Based on above architecture concepts the overall system architecture, its components and interactions are introduced next as well as an overall boundary delineation.

3.1 System Elements and Boundary

From a bird's eye view, three main systems exist that operate together in Triple Space Computing. One main system is TSC clients. A writer is a TSC client and so is a reader. In some cases, readers and writers are distinct, in others they might be the same. For representational purposes we do not distinguish them. In addition to this, another main system is a triple space server that can host any number of triple spaces. TSC clients communicate with the triple spaces (via the triple space server) by the Triple Space Transfer Protocol, TSTP, see Section 0. The following Figure 1 shows the situation graphically.

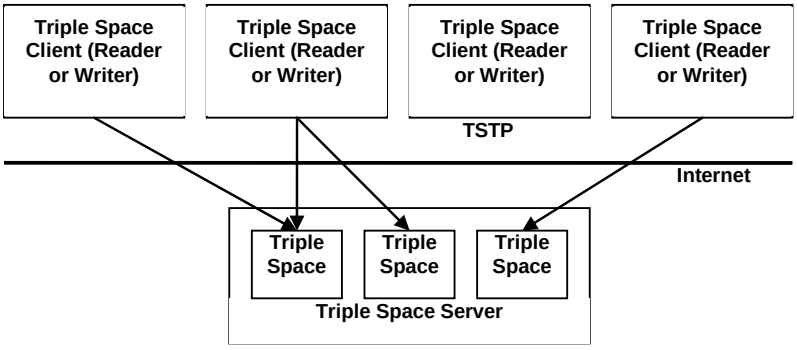

Figure 1: TSC System Elements and Boundaries

To further clarify the overall system layout, there can be several triple space servers on the Internet hosting hundreds of triple spaces[7]. Some clients might be connected to only one triple space, others to several triple spaces. Clients can be concurrently connected to several triple spaces, or in sequence, like clients of Web pages.

3.2 Triple Space Clients

A triple space client writes triples, reads triples or does both either at the same time or sequentially. Clients are therefore not distinguishable from the viewpoint of a triple space. Every client in general can read and write triples. The only way for a client to

[7] From a scaling perspective, every node in the Internet could have one (or more!) triple space servers installed!

access a triple space is through the TSTP (Triple Space Transfer Protocol). This protocol implements the read and the write operation on a triple space including all the necessary parameters.

3.3 Triple Space

A triple space is a virtual concept implemented by triple space servers. A triple space server implementation can implement this virtualization as it likes. Each triple space within a triple space server has to be distinguished so that written triples end up in the particular triple space the writer indicated when invoking the write operation.

A triple space server implementation, however, has some freedom to implement the virtualization. For example, one implementation might decide to implement the storage on the file system and it decides to put every triple in a separate file and create a directory for every triple space. Another implementation might decide to have a file for each triple space. Yet another implementation might decide to use a professional database system and implements a triple space as a relational database table.

In essence, a mapping has to happen from the read and write operation to the correct storage location in the persistent storage mechanism, be it files, databases or other storage mechanisms.

3.4 Triple Space Server

A triple space server hosts as many triple spaces as the owning organization decides. TSC clients do not know about triple space servers but only the virtual triple spaces. Consequently, management operations have to be available on a triple space server to create, delete and empty triple spaces:

- `create_triple_space (URL)`
  `returns` `success | triple_space_already_exists`
- `delete_triple_space (URL)`
  `returns` `success | triple_space_does_not_exist`
- `empty_triple_space (URL)`
  `returns` `success | triple_space_does_not_exist`

Convenience functions like listing existing triple spaces might be valuable. Also, it might be helpful to be able to prevent all access from a particular triple space in case problems are encountered or while storage space is added to the system environment. This might be achieved through an operation that puts a triple space into offline, i.e., making it inaccessible for clients.

3.5 Triple Space Transfer Protocol (TSTP)

The triple space transfer protocol is used between TSC clients and triple space servers to initiate the operations of writing and reading triples. A possible syntax for the two operations is

- `tstp://URL/triple`
- `tstp://URL/URI`

where ‘triple’ is assumed to have a URI as unique identifier as well as three values, subject, predicate and object. The syntax element ‘tstp’ is the protocol indicator. The first operation is the write operation where the URL of the triple space in question is provided followed by the triple to be written. The second operation is the read operation that provides the URL of the triple space and the URI of the triple to be read.

A possible implementation of this protocol is to define it in detail (preferably through a standards process, e.g. through W3C[8]) and expect all web server and application server vendors to implement it. However, an easier approach and alternative implementation is to map the TSTP protocol to the HTTP protocol. In this case there is no native implementation of it; however, it has the benefit of using a proven and Internet-scalable technology. This approach is suggested in Section 0.

3.6 Summary

In summary, the minimal architecture of TSC is small and simple. There are only a few components that have a quite simple interaction pattern structure. This allows making the components interact quite efficiently in an Internet-scaling implementation.

4 Implementation Suggestions

For the minimal TSC architecture to work properly an implementation for all the architecture concepts has to be provided. In the following some possibilities are discussed that an implementation might consider. Alternative approaches are possible, too. However, the suggestions were given with the idea of a minimal system in mind.

4.1 TSC Client

In the very extreme case, no implementation is provided at all for TSC clients. As already indicated, TSTP is piggybacked on the HTTP protocol with a syntax as defined in Section 0 (or a similar one). Therefore, any client that can invoke the HTTP protocol in principle can invoke the TSTP protocol, too.

However, for convenience, it might be good to have a Java class implemented that exposes the specific write and read operation of the TSTP protocol. In this case the software engineer who implements the client can use the Java class whenever TSTP invocations have to be performed. The Java class internally maps the parameter to the HTTP protocol as shown below and initiates the HTTP calls internally. Through this approach the software engineer does not have to worry about the TSTP to HTTP mapping at all and does not have to implement the correct HTTP syntax itself.

[8] www.w3c.org

4.2 TSC Server

The TSC server in general follows most likely a layered architecture. In a first approximation it has four components:

- **Storage component**. The storage component stores the triples. As storage component many alternatives are possible, including databases, file systems, RDF databases, persistent queues, etc.
- **HTTP communication component**. This component is the component that receives the HTTP calls that implement the TSTP protocol. Each invocation is either a write or a read. Each invocation is forwarded to the TSTP operation component.
- **TSTP operation component**. This component implements the functionality of the writing and reading of triples. It receives the write and read command from the HTTP communication component. In turn it invokes the appropriate TSC server operation and returns the result. While doing so, it ensures consistency, e.g., that the HTTP syntax is correct. It also does error handling. If e.g. a triple space is not existent, the TSC server returns a corresponding error message. In turn it composes an error code to be returned to the client as result of the HTTP invocation.
- **TSC server operations**. The triple space server finally implements the write and read operations for triples and does all the appropriate error handling. Furthermore, it implements the server operations for creating, deleting and emptying triple spaces.

Figure 2 shows the various components. As it is a layered architecture, code in upper layers call operations in lower layers. As can be seen, there are only a few components necessary and their invocation relationships are quite simple.

4.3 TSTP

The triple space transfer protocol (TSTP) is the protocol TSC clients use in order to write or to read triples from triple spaces. The implementation suggestion is to layer the TSTP protocol 'on top of' HTTP. This means in practical terms that a tstp protocol directive (like **`tstp://www.deri.org/phone_numbers/URI=102408`** requesting triple 102408) is translated to HTTP. The translation is simple and follows the following mapping:

- **`tstp://URL/triple`** is translated into **`http://URL/prot=tstp&uri=URI&s=<subject>&o=<object>&p=<predicate>`**. The URL is the same in both, however, in the HTTP translation one parameter (**`prot`**) states that this is a 'tstp' protocol invocation, one parameter (**`uri`**) contains the URI of the triple, and three parameters contain the values (**`s`**, **`o`** and **`p`**).
- **`tstp://URL/URI`** is translated into **`http://URL/prot=tstp&uri=URI`**. Here again **`prot`** indicates the protocol and since this is the read operation only the URI of the sought triple is provided.

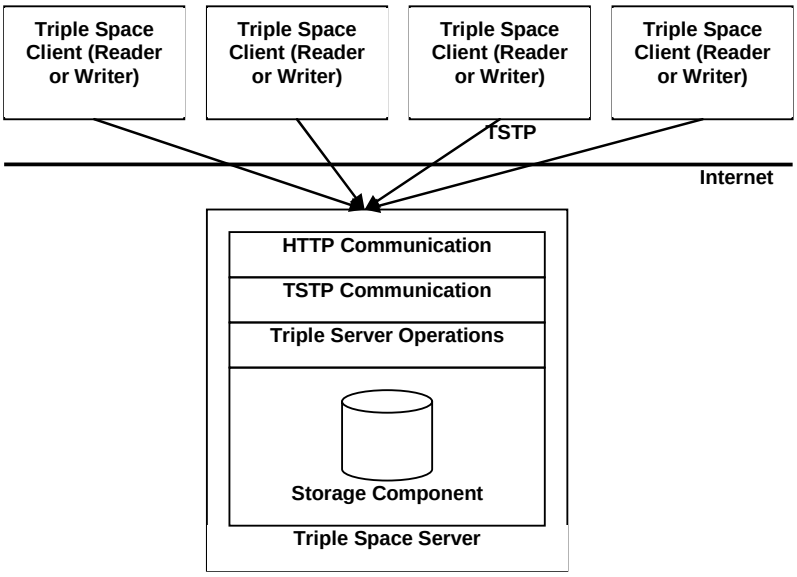

Figure 2: TSC Implementation Architecture

This translation from the TSTP protocol to the HTTP protocol is simple and can be done very fast as it is only a string rewrite, no external lookup of information is necessary at all.

4.4 Summary

In summary, this section has shown that a Internet-scale implementation of the TSC architecture is possible and will in fact work on Internet scale. From a comparison viewpoint the TSTP protocol and the HTTP protocol are scaling equally well as the TSTP protocol is mapped to the HTTP protocol. Furthermore, if professional storage software is used, the reading and writing of triples will be as fast as technology allows it to happen today.

5 Related Work

Related work in the space can be distinguished in base technology useful for implementing triple spaces and approaches with the same overall goal. Linda [Gelernter et al., 1985] is to some extent an early form of triple spaces. Linda follows the idea that processes can communicate by writing and reading from a shared storage location. Linda provides such a shared location as well as access operation for writing and reading data. The data model is tuples. Tuples can be written and read. Tuples do not

have an identifier like a URI, hence access to tuples is based on an associative approach. In order to access a tuple one or several fields have to be characterized by predicates and if a tuple can be found that has values matching the predicates, the tuple is returned.

Linda has a simple model, however, not based on Semantic Web technology. In addition, it is not based into the Web context, meaning that there is not the equivalent of the TSTP protocol at all. As a consequence, it is not Internet-scaleable. The concept of virtual triple spaces does not have an equivalent in Linda, either.

Semantic tuple spaces as described in [Khushraj et al., 2004] follow the idea to marry two technologies: semantic technology and tuple space technology. The approach is to use non-semantic technology (in this case JavaSpaces[9]) and require that one of the fields contains a semantically defined data item using OWL [McGuiness et al., 2004]. Fundamentally, a non-semantic implementation of tuple spaces was augmented with Semantic Web technology.

Aside from the fact that some additional features are implemented that go beyond those described here (and hence the proposed system is not minimal), other problems arise with this approach. No Web-scale access protocol like TSTP is defined and provided. No discussion takes place regarding the lost update problem. Virtualization of tuple spaces does not take place, and more fundamentally, the underlying data model is a mix of non-semantic and semantic technology (compared to clean semantic triples in the proposed minimal architecture). All in all, semantic tuple spaces point into the right direction, but do not have the potential for an Internet-scale system that is fully supporting Semantic Web technology.

RDF databases are on possible storage technology for the triple space architecture. RDF databases expose the operations necessary in order to retrieve and to store triples. Compared to the minimal triple space architecture they do not have a Web-wide access protocol like TSTP and are therefore not Web-wide accessible. Some RDF databases are Jena[10], Kowari[11], Redland[12], SESAME[13], or YARS[14]. Mainstream databases like Oracle plan to release RDF support very soon[15]. In addition, RDF databases do not implement the concept of virtual triple spaces; consequently they do not have the corresponding management functions. However, they are to be seriously considered as an implementation technology for the minimal triple space architecture.

In terms of additional features RDF databases might provide part of the required functionality already, or make it easy to implement the additional features. This has to be examined in detail when additional features going beyond the minimal architecture are designed and implemented. Databases that do not support RDF are not discussed as related work. Certainly it is possible to store RDF triples in a relational database, however, in this case databases would ‘only’ be an implementation technology.

---

[9] http://www.sun.com/software/jini/specs/js2_0.pdf
[10] http://jena.sourceforge.net/index.html
[11] http://www.kowari.org/
[12] http://librdf.org/
[13] http://www.openrdf.org/index.jsp
[14] http://sw.deri.org/2004/06/yars/yars.html
[15] http://www.w3.org/2005/Talks/0308-semweb-em/?n=18

Persistent queues are an asynchronous communication technology that also, depending on the particular implementation, can be persistent as well as secure. Queues in general implement FIFO access policies in the sense that one client appends (enqueues) a message to a queue and other clients dequeue messages from a queue. Example queuing systems are discussed in [Gray et al., 1993]. Examples of commercial implementations are AQ[16] and MQ-Series[17].

Aside from the particular access behavior, queues do not provide a simple Web access protocol like TSTP. In addition, the dequeue operation is destroying in the sense that if a message is dequeued, it disappears from the queue and cannot be read again. Also, the concept of a virtual triple space that is different from the triple server implementation does not exist. Like in the case of databases, a queuing system might be a very good implementation technology.

Interesting enough, FTP has a lightweight protocol for writing and reading files. The reading of a file is non-destroying, and FTP directories are Web compliant in the sense that once written they can be read any number of times. FTP is almost implementing a very rudimentary form of triple spaces. However, the storage object of FTP is files in general. Therefore, a FTP server cannot really check the format of data written within files. This means that a writer might write triples or anything else it wants. In addition, FTP does not have the concept of virtual triple spaces. In general, however, FTP could be an underlying technology instead of database technology. A lot of functionality would have to be added to it, but it could avoid the use of 'heavy' database technology.

6 Summary

The minimal triple space architecture as outlined in this article attempts to provide an asynchronous communication mechanism for machine-to-machine communication that supports the four types of autonomy: time, space, reference, and data schema. This new approach follows the Web-style of publishing information: persistent publish and read. A writer can persistently publish data and readers can read the data as often as they want, analogous to web pages. It was shown that it is possible to design an architecture that allows Internet-scale implementation of this new approach and an example from the Business-to-Business domain was given that demonstrated the tremendous benefits.

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT; by Science Foundation Ireland under the DERI-Lion project.

[16] http://www.oracle.com/technology/products/aq/index.html
[17] http://www.ibm.com/software/integration/wmq/

References

[Berners-Lee et al., 2005] T. Berners-Lee, R. Fielding, L. Masinter: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force (IETF), January 2005

[Boley et al., 2004] H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S. Tabet, G. Wagner: FOL RuleML: The First-Order Logic Web Language. Version History, 2004-11-02: Version 0.9, http://www.ruleml.org/fol/

[Bussler, 2003] C. Bussler: B2B Integration. Springer-Verlag, 2003

[Cabrera et al., 2004a] L. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey: Web Services Coordination (WS-Coordination), November 2004. ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf

[Cabrera et al., 2004b] L. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey, S. Thatte: Web Services Atomic Transaction (WS-AtomicTransaction), November 2004 . ftp://www6.software.ibm.com/software/developer/library/WS-AtomicTransaction.pdf

[Cabrera et al., 2004c] L. Cabrera, G. Copeland, T. Freund, J. Johnson, J. Klein, D. Langworthy, F. Leymann, D. Orchard, I. Robinson, T. Storey, S. Thatte: Web Services Business Activity Framework (WS-BusinessActivity), November 2004. ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf

[Christensen et al., 2001] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001, http://www.w3.org/TR/wsdl, March 2001

[Date, 2003] C. Date: An Introduction to Database Systems. Addison Wesley, 2003

[de Bruijn et al., 2005] J. de Bruijn, H. Lausen, R. Kummenacher, A. Polleres, L. Predoiu, M. Kifer, D. Fensel: D16.1v0.2 The Web Service Modeling Language WSML. WSML Final Draft 20 March 2005, http://www.wsmo.org/TR/d16/d16.1/v0.2/20050320/

[Fensel, 2004] D. Fensel: Triple-based Computing. DERI Research Report 2004-05-31, May 2004, http://www.deri.org/TR/2004-05-31, May 2004

[Fielding et al., 1999] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force (IETF), June 1999

[Gelerneter et al., 1985] D Gelernter, N. Carriero, S. Chang: Parallel Programming in Linda. Proceedings of the International Conference on Parallel Processing, 1985

[Gray et al., 1993] J. Gray, A. Reuter: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Series in Data Management Systems, 1993

[Horrocks et al., 2004] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[Khushraj et al., 2004] D. Khushraj, O. Lassila, T. Finin: sTuples: Semantic Tuple Spaces. Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), Boston, MA, USA, August 2004

[Manola et al., 2004] F. Manola, E. Miller (eds.): RDF Primer. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/, February 2004

[McGuiness et al., 2004] D. McGuineess, F. van Harmelen: OWL Web Ontology Language – Overview. W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-features/

[Mitra, 2003] N. Mitra (ed.): SOAP Version 1.2 Part 0: Primer. W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part0/, June 2003

[Pemberton et al., 2000] S. Pemberton, D. Austin, J. Axelsson, T. Çelik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navarro, S. Peruvemba, R. Relyea, S. Schnitzenbaumer, P. Stark: XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26 January 2000, revised 1 August 2002, http://www.w3.org/TR/xhtml1, January, 2000

[Rescorla et al., 1999] E. Rescorla, A. Schiffman, The Secure HyperText Transfer Protocol. RFC 2660, Internet Engineering Task Force (IETF), August 1999