

A Minimal Triple Space Computing Architecture

Christoph Bussler

April 2005

DERI Technical Report 2005-04-22

DERI Ireland
University Road
Galway
IRELAND
www.deri.ie

DERI Austria
Technikerstrasse 13
A-6020 Innsbruck
AUSTRIA
www.deri.at

A Minimal Triple Space Computing Architecture

Christoph Bussler

Digital Enterprise Research Institute (DERI)
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.

Table of Contents

1	Introduction to Triple Space Computing	5
1.1	Vision	5
1.2	Benefits	7
1.3	Goal of Paper and Outline	7
2	Requirements	8
2.1	Autonomy	8
2.2	Minimalism	8
2.3	Simple Functionality	9
2.4	Efficiency	9
2.5	Scalability	9
2.6	Dependability	9
2.7	Technology Originality	9
2.8	Summary	10
3	Architecture Concepts	10
3.1	Storage Object	10
3.2	Storage Location and Location Virtualization	11
3.3	Storage Operation	12
3.4	Storage Semantics	12
3.4.1	Write	12
3.4.2	Read	12
3.4.3	Concurrent Writes	13
3.4.4	Concurrent Reads	13
3.4.5	Concurrent Write and Read Operation Interleaving	13
3.5	Summary	13
4	Architecture Components and Interaction	14
4.1	System Elements and Boundary	14
4.2	Triple Space Clients	15
4.3	Triple Space	15
4.4	Triple Space Server	16
4.5	Triple Space Transfer Protocol (TSTP)	16
4.6	Summary	17
5	Implementation Suggestions	17
5.1	TSC Client	17
5.2	TSC Server	17
5.3	TSTP	19
5.4	Reader/Writer semantics	20
5.5	Summary	20
6	Beyond Minimalism	20
6.1	Search and Querying (Predicate-based Access)	20
6.2	Operation Semantics	21
6.3	Consistency Rules	22
6.4	Transaction	22
6.5	Reliability	22
6.6	Ontology Definition	23
6.7	Access Security	23

6.8	Transmission Security.....	23
6.9	Non-Repudiation.....	24
6.10	History and Archive.....	24
6.11	Set-oriented Operations.....	25
6.12	Graph-oriented Operations.....	25
6.13	Convenience Operations.....	25
6.14	Location Directory.....	26
6.15	Versioning.....	26
6.16	Summary.....	26
7	Example.....	26
7.1	Overview.....	27
7.2	Setup and Bootstrapping.....	27
7.3	Operation.....	27
8	Related Work.....	28
8.1	Tuple Spaces.....	28
8.1.1	Linda.....	28
8.1.2	Semantic Tuple Spaces.....	28
8.2	Technology.....	29
8.2.1	RDF Databases.....	29
8.2.2	Persistent Queues.....	29
8.2.3	FTP.....	30
9	Summary.....	30
10	Acknowledgements.....	30
11	References.....	30

1 Introduction to Triple Space Computing

The vision of Triple Space Computing is introduced based on a technology combination of Semantic Web technology with standard, off-the-shelf proven software technology¹. The benefits of the TSC approach are high-lighted and the specific goal of this article is introduced.

1.1 Vision

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]. This approach is clear when Web pages are analyzed. Anybody can define a Web page persistently and publish it at an Internet service provider or at a server at home or at work (this is usually done with help of Web servers). Publication means that the Web page is made accessible by anybody who knows the URL of that Web page and has a Web browser to display it². Persistent publication means that the Web page does not get lost in case of power or network failures. Once the power is back on or the network link is reestablished, the Web page is available again and unmodified compared to its content before the failure. Reading a Web page means to retrieve the Web page from the Web server by typing in the URL into a browser bar and requesting the display of 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 it³. The location for storing and accessing data is called triple space, and, as it will be later introduced, it is a virtual storage location.

Instead of following the ‘persistently publish and read’ paradigm, traditional Web Services based on the Web Service Definition Language (WSDL⁴, [Christensen et al., 2001]) and the Simple Access Object Protocol (SAOP⁵, [Mitra, 2003]) require a synchronous Hypertext-Transfer-Protocol (HTTP⁶, [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

¹ The name ‘Triple’ Space comes from RDF triples [Manola et al., 2004], as will be later introduced.

² E.g. <http://hometown.aol.com/chbussler>

³ 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.

⁴ <http://www.w3.org/TR/wsdl>

⁵ <http://www.w3.org/TR/soap/>

⁶ <http://www.ietf.org/rfc/rfc2616.txt>

read', this also prevents general mechanisms like Web caching and unique referencing of data through Uniform Resource Identifiers (URI⁷, [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 its internal storage (location autonomy), independent of the knowledge about potential readers (reference autonomy) and independent of its 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, in fact, 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.

One of the important goals is to bring TSC to an Internet-scale level. This means that any number of data providers and readers can write and read communication data across the whole Internet. Like in the case of Web pages, there should be no limitation in principle: the architecture is independent of system properties like response-time and throughput. The limitations that exist come only through the technology used. In order to become Internet-scalable, TSC are based on proven technology and their combination. As it will be discussed later in more detail later in this article, TSC will be based on the HTTP protocol, the Resource Description Framework (RDF⁸, [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.

⁷ <http://www.ietf.org/rfc/rfc3986.txt>

⁸ <http://www.w3.org/RDF/>

1.2 Benefits

The benefits of using TSC instead of synchronous communication for Web Service communication are manifold:

- **Web-style behavior.** The Web paradigm is well-known and therefore easy to understand and to use. This will allow setting up communication between communicating partners significantly more efficient.
- **Data semantics.** As the data is within the TSC infrastructure based on Semantic Web technology, the communication data is clearly semantically defined and hence machine understandable.
- **Asynchronous communication.** Asynchronous communication allows the data provider and reader autonomy. They can write and read according to their needs and independent from each other.
- **Reduction of uncertainty, delay and complexity.** Uncertainty of communication gets reduced as the writer of data can be ensured that data is persistently written if TSC does not return a fault message. The same applies for the reader. No more worries are necessary because of interrupted network communication. Communication delay is eliminated since the reader can read whenever necessary and the data provider can write whenever necessary. There is no need to wait for each other in order to enable communication synchronously.
- **Minimal impact in provider and reader's information systems.** There is no imposed impact to the writer and reader except accessing triple spaces through the triple space transfer protocol. It is as light-weight as HTTP itself (as will be shown later on).

In summary, there are quite a few benefits through TSC. In addition to the ones mentioned here, [Fensel, 2004] lists additional ones.

1.3 Goal of Paper and Outline

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' in order to establish a light-weight base line as light-weight as the analogous case of Web pages, Hypertext Markup Language (HTML⁹, [Pemberton et al., 2000]) and HTTP. So, the real goal – to stress it once more – is to define the

⁹ <http://www.w3.org/TR/html/>

simplest possible architecture and implementation model compared to ‘bells-and-whistles’ approaches as for example discussed in [Khushraj et al., 2004]¹⁰.

Section 2 enumerates the basic requirements for the minimal TSC architecture; Section 3 introduces the architecture concepts for it. Section 4 details the various architecture components necessary as well as their interactions from a dynamic perspective. Section 5 lists many implementation suggestions in order to utilize existing technology and knowledge to the extent possible. Section 6 introduces as many ‘bells-and-whistles’ as possible to show a whole array of possible future extensions that are interesting, but will come naturally with the cost of system complexity and system degradation in throughput and performance. Section 7 discusses briefly related work and Section 9 summarizes.

2 Requirements

Stating requirements is very important for the design of systems, be it their architecture or their implementation, in order to have guidance when choices have to be made. In the following the basic requirements are discussed that are relevant for the minimal TSC architecture and implementation.

2.1 Autonomy

As discussed earlier, there are four basic forms of autonomy that have to be realized by the architecture and its implementation: time, location, reference and data schema autonomy. Since those have been discussed already, they are not again discussed here.

2.2 Minimalism

It is the explicit goal to provide an as minimal as possible TSC architecture that is useful in its basic functionality. The reason for this minimalist approach is manifold, the main reasons are:

- **New research area.** TSC on an Internet-scale is a very recent and new research area. Before diving into possible functional and non-functional complexities (see Section 6) it is necessary to establish the base line of the minimal useful functionality. This will provide the lower boundary of functionality as a basis for experiments, exploration and continued research.
- **Simple existing systems.** The Web architecture as such is a very simple and yet (or because of this) a very powerful system. It has shown that simplicity and usefulness can go along extremely nicely with each other. Since TSC is proposing nothing else then the same paradigm for machine-to-machine communication, simplicity is a safe requirement to have. The goal is to make the minimal TSC

¹⁰ This is not meant to be a negative comment; instead, it is trying to make clear, especially when compared with Section 6, that the list of additional ‘useful’ feature appears to be almost endless.

architecture as simple as HTTP and HTML. Those made the web successful despite the possibility of broken links.

- **Viral technology.** In order for technology to be picked up to be useful on a wide basis it has to be simple and useful at the same time. Only then maximal impact can be achieved on an Internet-scale.
- **Pragmatics.** Finally, initial implementations usually work better when they are very simple. This is what the experience tells computer scientists and so this is a very good reason based on a pragmatic approach.

2.3 Simple Functionality

The minimal TSC architecture will only implement a very simple conceptual and functional model. In a nutshell, to make the minimal TSC a reality, triples can be written and read according to the four dimensions of autonomy. From a user viewpoint this simple model is immediately useful, easy to understand and use. From a system viewpoint this is easy to implement, to install and to maintain.

2.4 Efficiency

Efficiency is important. The architecture must be efficient, meaning few components and few essential interactions between them. The implementation, if efficient, is cost effective and easy to maintain. Both achieve also Internet-wide high-performance and high scalability.

2.5 Scalability

The requirement on scalability is that the minimal TSC architecture operates on Internet-scale like Web pages do already today. The requirement is therefore implying the use of proven Internet technology like HTTP, RDF as well as proven software technology like TCP/IP and databases (or alternative storage mechanisms).

2.6 Dependability

A minimal, simple, efficient, scalable system is worthless if it is not dependable. Dependable means in context of TSC that triple spaces are up and running most of the time (like Web pages) and that stored data does not get lost accidentally (like in working databases). Hence, the requirement is to make the architecture and implementation of TSC follow the model of the Web as well as to use dependable technology for it.

2.7 Technology Originality

A conceptually ‘cheap trick’ would be to mimic TSC as interface layer on non-semantic and non-Web technology like object management systems and remote procedure calls. While this would work from a technology point of view, it is not desirable in a (Semantic) Web context. Instead, the requirement is to natively use (Semantic) Web

technology and products that have this enabled as part of their functionality. Only then can TSC be part of the Web in native form.

2.8 Summary

In summary, all requirements for the minimal TSC architecture and implementation ask for a simple, yet powerful realization. The architecture concepts in Section 3, the components and their interactions in Section 4 and the implementation suggestions in Section 5 ensure that this main goal is reached.

3 Architecture Concepts

In many instances architecture is confused with graphical box diagrams and arcs between those. In these cases the concepts are not introduced at all, and so the diagrams often have a random and fuzzy meaning. The approach taken here is different. 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.

3.1 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. [Manola et al., 2004] uses the example of the creation date of a web page, see Figure 1. In this example the subject is ‘`http://www.example.org/index.html`’, the object is ‘August 16, 1999’ and the predicate is ‘`http://www.example.org/terms/creation-date`’.

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.     xmlns:extermns="http://www.example.org/terms/">
4.     <rdf:Description rdf:about="http://www.example.org/index.html">
5.         <extermns:creation-date>August 16, 1999</extermns:creation-date>
6.     </rdf:Description>
7. </rdf:RDF>
```

Figure 1: Example of RDF Triple

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 architecture. In an advanced version it might be possible to add the functionality of retrieving all triples without knowing their URI beforehand, see Section 6.1.

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. Again, more elaborate versions of the triple space might very well consider providing support for this case, see Section 6.12.

For a reader, however, this is not a problem. Once a reader read 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.

3.2 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 implement of ‘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.

3.3 Storage Operation

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 4.4.

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.

Both operations are sufficient for a minimal TSC architecture to work properly and be sufficiently useful. The next section will define their operational semantics in more detail.

3.4 Storage Semantics

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

3.4.1 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.

3.4.2 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

3.4.3 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.

3.4.4 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.

3.4.5 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 than 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.

3.5 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.

4 Architecture Components and Interaction

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

4.1 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 4.5. The following Figure 2 shows the situation graphically.

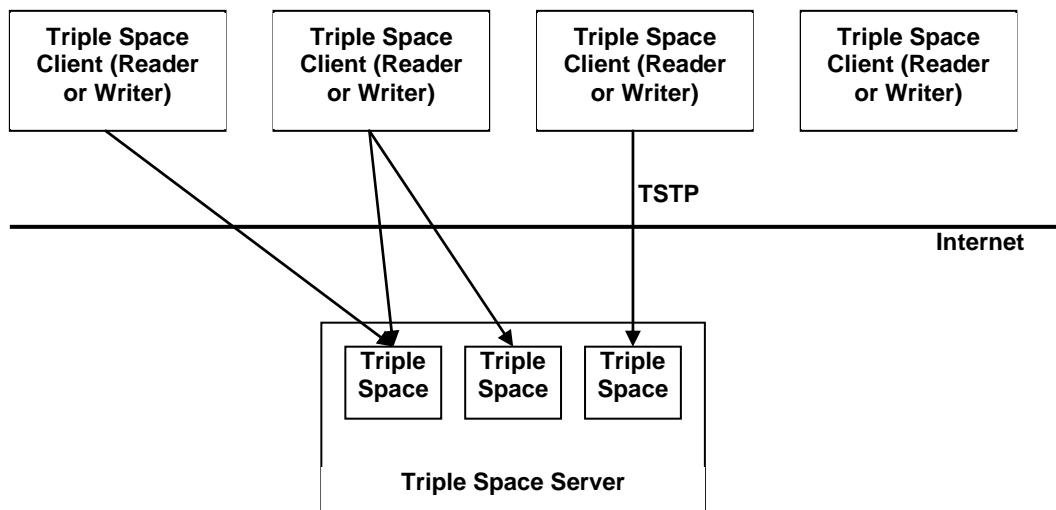


Figure 2: TSC System Elements and Boundaries

To further clarify the overall system layout, there can be several triple space servers on the Internet hosting hundreds of triple spaces¹¹. 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. Figure 3 depicts the situation graphically.

¹¹ From a scaling perspective, every node in the Internet could have one (or more!) triple space servers installed!

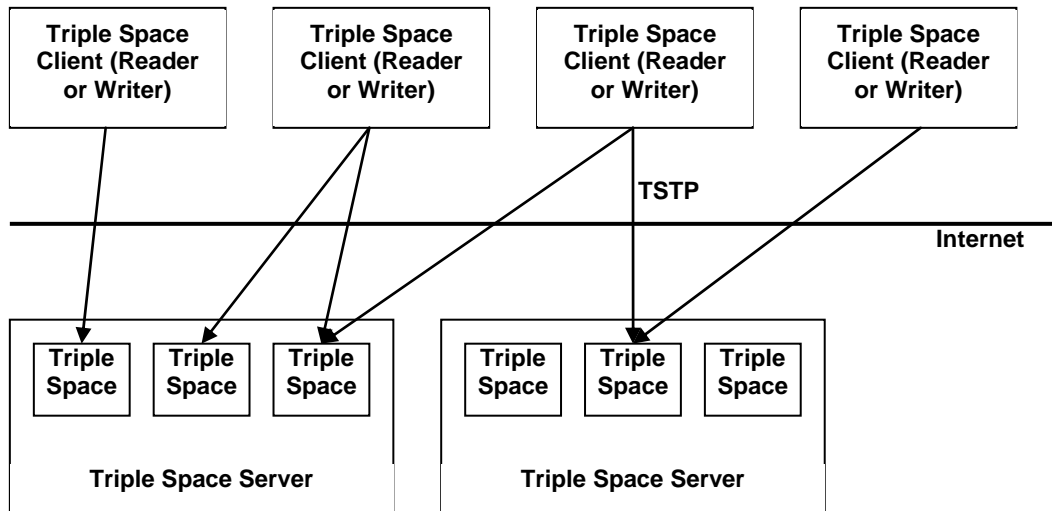


Figure 3: TSC with several triple space servers and clients

After this overview discussion the various elements are introduced in more detail individually next.

4.2 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 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.

4.3 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 decides 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.

4.4 Triple Space Server

A triple space server hosts as many triple spaces as the 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.

4.5 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 W3C¹²) 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 5.3.

4.6 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.

5 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.

5.1 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 5.3 (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.

5.2 TSC Server

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

¹² www.w3c.org

- **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 4 shows the various components. As it is a layered architecture, upper layers call lower layers. As can be seen, there are only a few components necessary and their invocation relationships are quite simple.

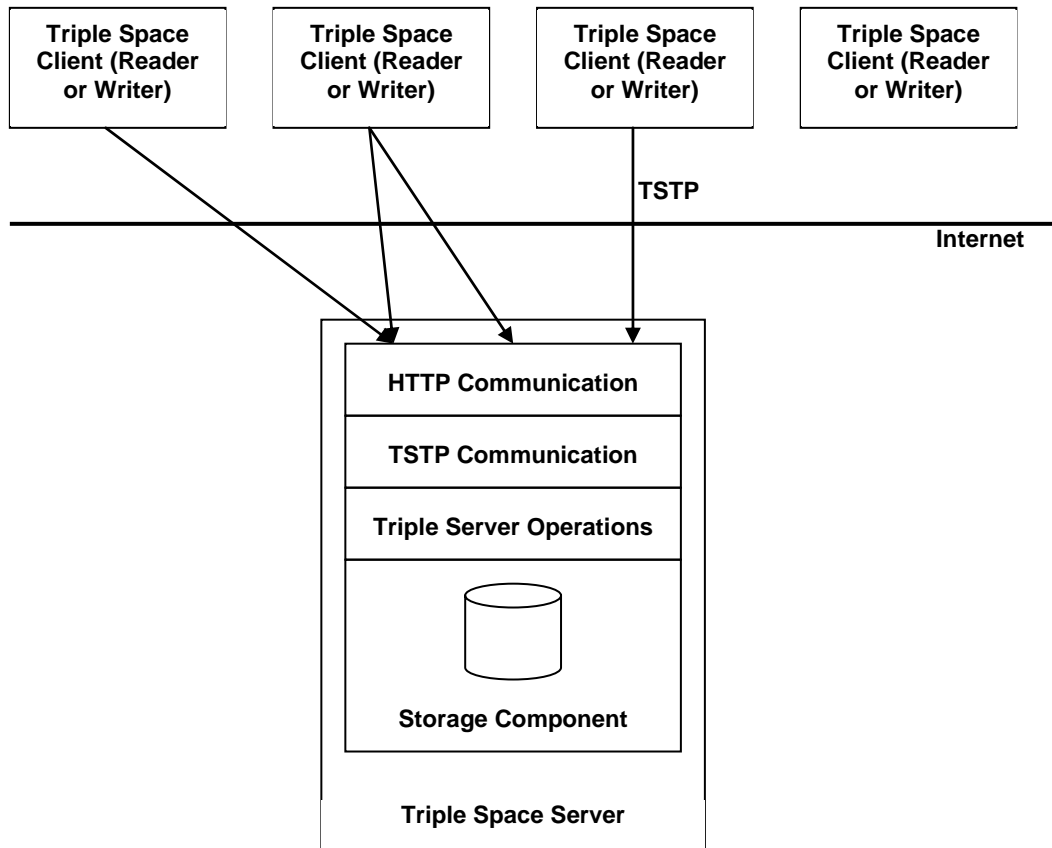


Figure 4: TSC Implementation Architecture

5.3 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.

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.

5.4 Reader/Writer semantics

The reader/writer problem is a well-known problem in Computer Science and does not require a lengthy discussion at this point. However, its implementation depends also on the underlying storage mechanism used. For example, when a primitive file system is used, no support for the reader/writer problem can be expected and the TSC server implementation has to take care of it itself.

However, if more sophisticated storage mechanisms are used like e.g. relational databases, then the transaction subsystem of those can be used. If all operations on the TSC server are guarded under transaction control, the reader/writer semantics is fully addressed.

5.5 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.

6 Beyond Minimalism

Of course, it is tempting to immediately respond to the minimal TSC architecture with the reaction that it is too simple and it is hard to believe that it will ever going to be useful. Therefore, in order to show the impact of this reaction, in the following additional potential features are discussed briefly; the order of the discussion is not significant. The risk of adding those features is to loose the simplicity as well as the performance of the minimal TSC architecture.

Real applications over time might require more functionality then what the minimal TSC architecture provides. The various applications of the TSC architecture will over time determine the necessary features as well as the nice-to-have ones.

6.1 Search and Querying (Predicate-based Access)

Reading triples based on the knowledge of their URIs only is not satisfying for more complex retrieval patterns. Instead of providing a URI to retrieve triples, a predicate might be more useful. For example, in order to get all phone numbers of the Semantic

Web Service cluster of DERI¹³, a predicate would be provided instead of the URI: `tstp://www.deri.org/phone_numbers/pred=<phone numbers of members of SWS cluster>`.

Of course, in a real implementation, a real query syntax must be implemented. In addition, set-oriented operations are necessary since a predicate identifies several triples in the general case (see Section 6.11) and the set of those has to be returned.

6.2 Operation Semantics

In the minimal architecture, once a triple is written, it can be read any number of times and its values can be changed any number of times. However, for specific application scenarios additional operation semantics might be desirable. In the following, examples are provided of useful operation semantics.

- **Destroying read.** One additional behavior is that the first reader of a triple destroys it, i.e., the triple does not exist afterward any more. This implements a single consumer queue. One or several writer can change the value of the triple, until the first reader reads it. Afterwards, the triple is non-existent and additional read or write attempts fail.
- **Addressed write.** A writer of a triple might define one or more readers that are eligible for reading the triple. When the last of the readers read the triple, it is destroyed. If a non-named reader tries to read the triple, a failure occurs. This behavior is ensuring that triple are only read by the right addressees. This is an important feature for e-Commerce where a RFQ [Bussler, 2003] is only sent to specific pre-selected suppliers. From a queuing perspective this implements a multi-consumer queue whereby the consumers are named.
- **Modified addressed write.** Alternative addressed write behavior exists. The destroying of the triple might occur after a subset of the named readers read the triple. Or, instead of the writer naming the readers, the writer might provide a predicate over readers and each reader qualifying will be allowed to read the triple.
- **Time-based destruction.** Triples might have a defined lifetime and after its expiration they are destroyed, being read once or multiple times or not.
- **Access-number based destruction.** Destroying of a triple might happen after some predefined number of reads.

As can be seen, it is easily possible to define interesting and desirable variation in the functionality and semantics of the simple write and read operation. However, the default is non-destroying behavior.

¹³ <http://sws.deri.ie/>

In a general implementation, the various semantics of the operations might be dynamic on the one hand side, and dynamically configurable on the other hand side. Dynamic semantics of operations means that a TSC server has a management interface that allows the modification and addition of additional semantics. Dynamic configuration means that the semantics can be defined on a per triple basis or a per triple space basis. This allows each writer to state the behavior it expects for its written triples.

6.3 Consistency Rules

Between triples consistency rules can be defined. For example, a triple containing the name of a city and a related triple containing a zip code might be constraint by the rule that the city and the zip code must correspond. Another example might be that each member of DERI must have a home page with a non-broken URL.

Consistency rules constrain the values in triples. Instead of the clients enforcing the correct values, it can be done by the TSC server. In this case each write will only be successful if the consistency rules that apply are obeyed. This is related to enforced database constraints in relational database systems [Date, 2003].

Various semantic Web rules languages are currently in the research discussion (e.g. [de Bruijn et al., 2005] [Boley et al., 2004] [Horrocks et al., 2004]) and might be all useful in a TSC server implementation.

6.4 Transaction

The read and write operations implement the reader/writer problem so that no lost updates occur. However, in the minimal architecture, this is done on a per triple basis. If sets of triples have to be written or read the lost update problem can occur on the set of triples as such. In this case the only solution is transaction protected set-oriented operations. Going beyond this case, transactions are necessary if several set-oriented operations should be protected from lost updates. In this case dedicated transaction operations are necessary that allow clients to delineate the transaction boundaries.

Since from a transaction perspective TSC servers are resource managers, immediately the question of distributed transaction control appears as the state of clients and the TSC server(s) in question might have to be coordinated. Web Service transaction protocols like the one discussed in [Cabrera et al., 2004a], [Cabrera et al., 2004b] and [Cabrera et al., 2004c] can be of invaluable help here. These would allow having clients and TSC servers be coordinated by transactions over the Internet.

6.5 Reliability

The minimal TSC architecture is reliable following the model of Web pages. The hosting TSC server of the triples stored in triple spaces has to ensure that existing triples are fully restored after system or network failures without any loss of data.

However, for some application scenarios this might not be enough. These might require a dependable access to triples despite of system failures or network failures. In this case alternative paths to the TSC server are helpful as well as mirrored systems.

Another step in this direction (that also addresses availability) is the replication of triples in the TSC server infrastructure or even across TSC servers.

6.6 Ontology Definition

Triples refer to values in their subject, predicate and object fields. These values are instances of concepts or relationship, in ontology speak. In order to increase the semantics in TSC it might be valuable to allow TSC server to host ontology definitions that are used for modeling the definitions of triples. In this case the values in triples must follow the ontology definition and a uniform semantically defined triple set can be ensured.

6.7 Access Security

As already indicated in the discussion about the semantics of triple space operations not every reader must necessarily be able to retrieve all triples. If a writer cares about being selective in the set of readers who should see his written triples, a writer might impose access control on his triples so that only authorized readers can access the triples.

The opposite case exists, too. Not all triple spaces might allow all writers to write triples. Denial-of-resource attacks might be abundant once an unprotected triple space URL becomes known. Even a triple space crawler can be envisioned that tries to detect triple spaces for malicious intents (or non-malicious ones). A triple space or even a TSC server can impose write access control in order to only allow authorized writers to write triples into the triple spaces.

6.8 Transmission Security

One ‘feature’ of the Web is its transparency: it is open for everybody and not a real secure space with tight access controls. Hence, if data is sent across it the sender has to ensure secrecy if it does not want to have anybody read it. The same applies for the write and read operation of triple spaces. The writer or reader, if they do not want to have anybody see the data they transfer, must ensure that nobody can read it while it is in transit from the triple space to them or vice versa.

One mechanism to achieve this is to either encrypt the data being transferred or the communication channel itself. Latter can be achieved with HTTPS [Rescorla et al., 1999], the Secure Hypertext Transfer Protocol¹⁴. In this case the data itself is in clear text; however, the communication channel ensures encryption while the data is in transfer.

¹⁴ <http://www.ietf.org/rfc/rfc2660.txt>

Alternatively, the data itself is encrypted. In this case everybody can access the data, but because of the encryption, nobody can read it. Obviously, the receiver should be able to read it. Therefore, using the mechanism of public and private key encryption it is possible that only the receiver can decrypt the data to interpretable syntax [Bussler, 2003].

The benefit of encrypting the communication channel is that the triples can be stored in the triple space in clear text allowing to search by their content. If they were stored encrypted, any access requiring to read the content would be impossible (e.g., search).

6.9 Non-Repudiation

Sometimes a dispute arises between the sender and receiver of e-commerce message as to the precise content of the messages. A sender might state that it ordered 1000 units of a product whereas the receiver claims only 100 units were ordered. Both can refer to the copies of their messages (the one sent and the one received), but it is not clear in this case how the '0' had gone missing, either by accident or by tampering with the content.

To solve this situation a mechanism must be put in place that allows both, the sender and receiver, to proof that the message they hold in hands is the originally sent or received one. One mechanism for that is to involve a third party that will hold a copy of the message, too. In a case of dispute between sender and receiver, the third party's copy will be the determining factor, i.e., it will be the normative copy that will be the basis of any decision.

Triple spaces can be the third party, given that once a triple is written it cannot be altered any more.

6.10 History and Archive

Writing and reading triples takes place over time by different triple space clients. Unless a specific provision is made, the precise order, time and contents of when triples are read or written will not be stored by any triple space as this information is not recorded at all. Furthermore, triples can be overwritten many times, and in an enhanced version possibly deleted.

Therefore, a triple space implementation can provide a history functionality that ensures that every 'movement' of triples, be it writing them, reading them, deleting them or updating them is recorded as history. At any point in time the history can make a precise statement about the ongoing operations.

While it is certainly desirable to have the complete history accessible all the time, the sheer data volume might not allow this as the storage space might be insufficient. Hence, at certain intervals, the history might be put into an archive that allows the reconstruction of the history on demand.

6.11 Set-oriented Operations

If a triple space client has to write many triples it would be more convenient for the client to write them all at once instead of calling the write operation several times. A write operation extension to allow set-oriented write would be convenient:

- write (set_of (triple))

At the same time, clients might read with a predicate as parameter, see Section 6.1. In this case the result might be very well a set of triples, not just one (or none) in the general case. So the read operation should be able to return sets of triples:

- read (predicate) **returns** set_of (triple) | error_code

Set-oriented operations are convenient and important at the same time and certainly a useful extension of the triple space architecture.

6.12 Graph-oriented Operations

Triples by their nature represent graphs in the general case. Triples can refer to each other to represent complex structures in form of graphs. If a client has a graph and wants to write it to a triple space or if a client wants to retrieve a graph from a triple space, graph-oriented operations would be convenient. Instead of the client converting a triple graph into a set of triples before writing it or a client retrieving a set of triples and having to convert this set into a triple graph it would be helpful to have graph-oriented operations available.

For writing triple graphs held by the client this means that on the client a convenience operation must be available that takes in a triple graph and produces a set of triples from that. Then, this would invoke the set-oriented operation of the TSTP protocol. This means that the conversion of a triple graph into the corresponding set of triples would happen at the client itself.

Reading a graph works slightly different. When a client wants to read a graph, it has to identify the triple that represents the root of the graph. The triple space then determines the graph starting from this root triple. Once the triple graph is determined, the corresponding set of triples is generated and returned as a set to the client. At the client, the set is converted into a corresponding triple graph again.

6.13 Convenience Operations

Being able to read and to write triples is the base functionality required. However, convenience operations like update and delete are important, too. An update operation would update an identified triple with new values. A delete operation would delete a triple altogether.

6.14 Location Directory

Triple spaces are uniquely identified by URLs. Like on the Web, once a URL is known, it is possible to access it. However, if a URL is not known, the only way to find a web page is through asking colleagues for the URL or use a search engine with the hope that the URL can be retrieved.

Triple spaces will have similar problems, i.e., only if the URL of a triple space is known, it can be accessed. If the existence of a triple space is known, but not its URL, then there is no way to access it.

A triple space search engine operating Web-wide is one possibility. It would attempt to find all triple spaces available and list them. Possibly it would access the triples themselves and display possibly a content summary in order to identify a theme (if possible at all) of the triple space.

Alternatively (or in addition) a triple space directory is feasible where triple spaces are registered in case they should be publicly known. A triple space directory would be used by triple space clients to find appropriate triple spaces for interaction with them.

6.15 Versioning

In some applications it is necessary to keep old versions of triples in order to keep the various values. This requires versioning support of triples in the triple space. An update of a triple in the triple space in this case would create a new version of the triple instead of overwriting its values. The versions of the triple are kept this way in the triple space ensuring that all prior values of the triples are available later on. At any point in time it is possible to retrieve all prior versions for inspection.

6.16 Summary

In summary, additional features are abundant and one can feel like a kid in a toy store with all these nice things around. However, caution is to be commended as it is possible to overshoot the functionality and in no time the triple space implementation turns into the most powerful distributed object management system ever with only a little likelihood for success.

7 Example

At this point triple spaces have been introduced in an abstract way as a very simple and powerful mechanism for Web-wide asynchronous communication. An example is in order here and it will be from the Business-to-Business integration domain [Bussler, 2003].

7.1 Overview

A very often found scenario is two enterprises in a buyer-seller relationship. The buyer buys from the seller goods and in order to initiate the process sends a purchase order (PO). The seller acknowledges that and returns a purchase order acknowledgment (POA). This simple example is enough to show the benefits of triple spaces. In reality, the purchasing process is a lot more complex and involved, including request for quotations, invoices, shipments, payments, and so on.

7.2 Setup and Bootstrapping

In order for the buyer and seller to interact, several initial setup and bootstrap steps are necessary:

- A triple space has to be created
- The address of it has to be communicated

A triple space does not float 'out there' in the Web. It has to be located at an organization that can host triple space servers. In the case of the example the seller decides to provide triple spaces for both, the buyer and seller to communicate. One triple space is designated for the POs, the other for the POAs. The URLs are www.seller.com/incoming_POs and www.seller.com/outgoing_POAs.

The buyer writes the POs into the [incoming_POs](http://www.seller.com/incoming_POs) triple space and reads the POAs from the [outgoing_POAs](http://www.seller.com/outgoing_POAs) triple space.

Once the seller has set up the two triple spaces, it sends the URLs to the buyer for the buyer to know the URLs. This sending is in this case done again using Internet technology, the e-mail system.

After this exchange of the triple space URLs, both seller and buyer can interact through the triple spaces. Nothing more is needed at all at this time.

7.3 Operation

One the buyer decided to buy goods from the seller it inserts a PO into the www.seller.com/incoming_POs triple space. The purchase order is represented as triples. Once the PO is written, the buyer is waiting for the POA. It does so by repeatedly accessing the www.seller.com/outgoing_POAs triple space looking for a corresponding POA.

The seller, once it receives the PO, does the internal operations finding out if it can fulfill the PO and if so, writes a corresponding POA into www.seller.com/outgoing_POAs.

As can be seen, both, the buyer and seller can operate without being both online at the same time (time autonomy). They do not have to share memory as both using triple spaces for writing and reading their data (location autonomy). They do not have to know

each other directly, both only refer to the triple space (reference autonomy). And, finally, both write triples independent of their internal representation of POs and POAs (schema autonomy).

While being a small example, it already can show the power of the minimal triple space architecture based on the four types of autonomy: time, location, reference and schema.

8 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. In the following they are labeled ‘technology’ and ‘tuple spaces’.

8.1 Tuple Spaces

8.1.1 Linda

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.

8.1.2 Semantic Tuple Spaces

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 JavaSpaces¹⁵) and require that one of the fields contains a semantically defined data item using OWL [McGuinness et al., 2004]. Fundamentally, a non-semantic implementation of tuple spaces was augmented with Semantic Web technology.

Aside from the fact that some of the additional features as discussed in Section 6 have been implemented (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

¹⁵ http://www.sun.com/software/jini/specs/js2_0.pdf

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.

8.2 Technology

8.2.1 RDF Databases

RDF databases are one 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 Jena¹⁶, SESAME¹⁷, or YARS¹⁸. Mainstream databases like Oracle plan to release RDF support very soon¹⁹.

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.

8.2.2 Persistent Queues

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 AQ²⁰ and MQ-Series²¹.

¹⁶ <http://jena.sourceforge.net/index.html>

¹⁷ <http://www.openrdf.org/index.jsp>

¹⁸ <http://sw.deri.org/2004/06/yars/yars.html>

¹⁹ <http://www.w3.org/2005/Talks/0308-semweb-em/?n=18>

²⁰ <http://www.oracle.com/technology/products/aq/index.html>

²¹ <http://www.ibm.com/software/integration/wmq/>

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, especially when particular access policies are to be implemented (see Section 6.2).

8.2.3 FTP

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.

9 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.

10 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.

11 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. Grosz, 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. Kummener, 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
- [Gelernter 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. Grosz, 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

[McGuinness et al., 2004] D. McGuinness, 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