WWW
or
What Is Wrong with Web Services

Reto Krummenacher[1], Martin Hepp[1], Axel Polleres[1], Christoph Bussler[2] and Dieter Fensel[1,2]

Digital Enterprise Research Institute
[1] University of Innsbruck, Austria
[2] National University of Ireland, Galway, Ireland

Abstract

A core paradigm of the Web is information exchange via persistent publication, i.e., one party publishes a piece of information on the Web, and any other party who knows the location of the resource can retrieve and process the information at any later point in time and without the need for synchronization with the original publisher. This functionality significantly contributed to the scalability of the Web, since it reduced the amount of interaction between the sender and the recipient. Current approaches of extending the World Wide Web from a collection of human-readable information, connecting humans, into a network that connects computing devices based on machine-processable semantics of data lack this feature and are instead based on tightly-coupled message exchange. In this paper, we (1) show that Web services based on the message-exchange paradigm are not fully compliant with core paradigms of the Web itself, (2) outline how the idea of persistent publication as a communication paradigm can be beneficially applied to Web services, and (3) propose a minimal architecture for fully Web-enabled Semantic Web services based on publication in shared information spaces, which we call Triple Space Computing.

1. Introduction

The ***World Wide Web*** is a tremendous success story, both in terms of the amount of available information and in terms of the growth rate of the number of human users. Starting as a closed network for exchanging scientific information, it has become a global media used for information dissemination and information access within slightly more than a decade. In many respects the Web has become the major means for publishing and accessing information, and it is

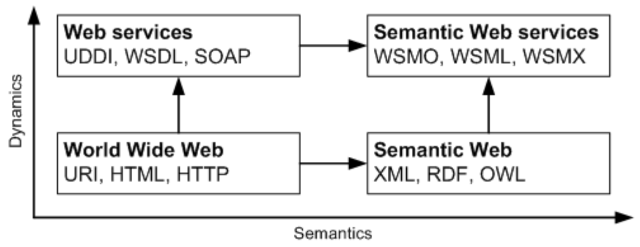

Figure 1. The four major stages in the development of the Web

expected to connect one billion people in only a few years. Its scalability, simplicity and the comfort and speed in disseminating information is unique. However, it is solely a network for humans. Computers do not have access to the actual meaning of the content and in return only provide very little support in accessing and processing this information. There are two complementary trends that are about to change this fact and aim at transforming the Web from being for humans only into a network that involves computers in order to provide support for human interaction at a much higher level than possible with current Web technology (see Fig. 1):

- The ***Semantic Web*** adds machine-processable semantics to data [2].
- ***Web services*** provide the platform-neutral integration of arbitrary applications [1].

Eventually, ***Semantic Web services*** (the combination of Semantic Web and Web service technology) promise a fully automated Web in which functionality can be accessed by machines, by adding machine-processable data to the description of Web services. Semantic Web services aim at allowing mechanized service discovery, service configuration, combination (i.e. realizing complex workflows and business logics with Web services) and service comparison, as well as automated negotiation [7].

Currently, the available (non-semantic) Web service technologies are far from providing the desired state of automated, seamless integration. The obstacles may eventually be overcome. It is possible, however, that these unsolved issues are an indication for a more fundamental problem with the current direction of Web services technology. Web services are mainly based on transient message exchange and not on persistent publication of data. As a matter of fact, Web services are far from using the Web as means for information publication and access. Instead of following the 'persistently publish and read' paradigm of the Web, traditional Web services using WSDL [6] and SOAP [18] establish a tightly coupled communication cycle, most frequently using a synchronous HTTP transaction to transmit data. URIs, which are meant as unique and persistent identifiers for resources are used only for the identification of the participant, whereas the information exchanged is hidden in the SOAP message.

In this paper, we address the current misfit between the fundamental principles of the Web and the current direction of Web services. First of all, we analyze current Web service technology in Section 2 and demonstrate that current Web services do not have much in common with the Web. After questioning the term "Web" in "Web services", we show in Section 3 how true Web services could be implemented based on persistent publication in shared information spaces. The foundation is to augment space-based computing with machine-accessible semantics, and to apply this idea to Semantic Web services. We show how, by adding semantics directly to this alternative communication paradigm, the resulting space-based infrastructure allows for fully machine-enabled, asynchronous data interchange. In Section 4 we outline a minimal Internet-scale architecture for Triple Space Computing. Section 5 sketches how trust and security issues could be addressed in a Triple Space. Finally, Section 6 discusses related efforts and Section 7 concludes the paper.

2. Are Web services really Web services?

In this section we argue that, besides their name, Web services do not have much in common with the Web. Current Web service technology is based on tightly coupled message exchange. This is similar to the situation in the pre-Web age, where one had to send an e-mail message to a scientific colleague, asking for a specific paper or piece of information. In the current Web as an infrastructure for humans, this pattern of communication has widely been replaced by persistent publication and asynchronous retrieval. We no longer request the biggest share of information in lengthy, synchronous communication cycles with the originators, but fetch it from persistent sources, e.g. the scientist's personal Web page. We argue that this has significantly contributed to the success and scalability of the Web, since it freed the sender from the need to handle individual requests, and it made resources identifiable and thus referable. Imagine you would have to handle individual requests from all your scholarly colleagues who visit your Web site. You might soon become the main bottleneck for the dissemination of your own ideas. The shift from information dissemination based on message exchange not only made the Web scale tremendously, it has also sped up the dissemination process. When comparing Web services with this important principle of the Web it becomes very obvious that Web services are not following the core idea of the Web. It is true that Web services use the Internet as the transport medium. However, that is more or less all they have in common with the Web. The idea of publishing data and accessing it asynchronously is lacking. Instead of publishing the information based on a global and persistent URI, Web services establish (1) stateful conversations based on the (2) hidden content of messages.

The negative effect of such distributed applications that communicate via message exchange is that they require a strong coupling in terms of reference and time. This means that traditional Web services require that the sender and receiver of data (1) maintain a connection at the very same time, that they (2) agree on the data format, and (3) know each other and share a common representation. Therefore, the communication has to be directed to a particular service and is synchronous as long as neither party implements asynchronous communication (and jointly agree on the specific way this mechanism is implemented).

The above statement is in alignment with the worries expressed by the REST community [9]. Their two major criticisms around Web services are about the improper usage of URIs and the violation of the stateless architecture of the Web. It is one of the basic design principles of the Web and REST architecture to not provide stateful protocols and resources [10], [21].

In practice this means that when sending and receiving SOAP messages, the content of the information is hidden in the body and not addressed as an explicit Web resource with its own URI. Therefore, most Web functionality dealing with caching or security checks is disabled, since using it would require parsing and understanding all possible XML schemas that can be used to write a SOAP message. Thus, when a stateful conversation is required, this should be explicitly modelled by different URIs. Moreover, referring to the content transmitted via an explicit URI in an HTTP-request would allow the content of a message to be treated like any other Web resource.

In the next section we will explore how Web services can be fully based on the very successful ”persistently publish and read” Web principle and propose means to overcome the major flaw that recent Web services suffer from.

3. Triple Space computing

In this section we present Triple Space computing (TSC, [8]) as an approach to address the widely-recognized problems with Web services. We start with discussing Tuple Space computing as a paradigm for exchanging data between processes that is very similar to the paradigms of the Web. We introduce the concept of semantic self-description of information which naturally leads us to what we call the Triple Space.

3.1. Tuple Space computing

Tuple Spaces were introduced in parallel programming languages such as Linda to implement communication between parallel processes [12]. Instead of sending messages back and forth, a simple communication interface is provided, and processes can write, delete, and read tuples from a persistent space. A tuple is an ordered set of typed fields, each of which either contains a value or is undefined, while the Tuple Space is an abstract space containing tuples and is visible to all processes. The tuples can be read from the space by use of templates. A template matches tuples that have the same internal structure, i.e., the same number and type of fields. Tuple Space computing has very strong advantages. It naturally decouples processes along three orthogonal dimensions:

- **reference**. No need to address communication partners directly.
- **time**. No need for synchronous links between communication partners.
- **space**. No need to run in the same computational environment as long as access to the same space is guaranteed.

This decoupling has obvious design advantages for defining reusable, distributed, heterogeneous, and agile communicating applications. A communication paradigm based on Tuple Space computing also resembles a core Web paradigm: information is persistently written to a globally accessible space where it can easily be accessed without a cascade of message exchange operations. It is thus quite obvious that the Web and Tuple Spaces share many underlying principles. However, [14] report on several shortcomings. Current Tuple Space models lack support for namespaces, semantics, and structure in describing the information content of the tuples. Part of these limitations are addressed by existing Web technology for resource identification in the Web:

1. URIs provide a reference mechanism that allows information to be distinguished on a world-wide scale.
2. Namespaces provide a separation mechanism that allows different applications to use the same vocabulary without blurring their communication.

Tuple Spaces provide a flat and simple data model that does not support any nesting or linking of data. Hence, tuples with the same number of fields and field order but different semantics cannot be distinguished. In the following we propose to extend Tuple Space computing into what we call Triple Space computing to enhance the common field structure and field comparison matching with nested and interlinked tuples (triples) and semantics-aware matching rules.

3.2. Triple Space computing

Fortunately, the Resource Description Framework (RDF,[16]) already provides the fundamentals for a shared information space that provides some degree of machine-access. In our opinion, RDF is the natural link from the space-based computing paradigm into the Semantic Web. Notice that the Semantic Web will not become obsolete by the Triple Space paradigm, nor do we assume that message exchange will completely disappear as a communication paradigm for accessing computer functionality remotely. Information spaces can help overcome heterogeneity in communication and cooperation, however, they do not provide any answer to data heterogeneity. That remains a task for the Semantic Web, as it provides standards to represent machine-processable semantics of data. In RDF, semantic data is described by interlinked triples of the form <subject,predicate,object>. The subject is the resource about which the statement is made. The predicate represents a specific property and the object is either a resource or a literal defining the property value of the statement. Resources are identified by URIs, while literals are e.g. strings or numbers. RDF Schema [4] additionally defines classes, properties, domain and range inferences, and hierarchies of classes and properties on top of RDF. Therefore, a richer data model than interlinked triples could also be used to model and retrieve information. This gets even further extended by OWL [17], a data modelling language based on description logic.

The Semantic Web aims at making problems in protocol and process heterogeneity transparent by its uniform and simple means for accessing and retrieving information. Space-based computing moreover allows complex message exchanges to be replaced by simple read and write operations in a globally accessible space. Therefore, Triple

Spaces have the potential to become a global platform for application and service integration on the Internet just as Tuple Space computing became a means for the local integration of parallel processes.

The Triple Space shall offer an infrastructure that scales conceptually on an Internet level. Like Web servers publish Web pages for humans to read, Triple Space servers would provide Triple Spaces to publish machine-interpretable data. Providers and consumers could publish and consume triples over a globally accessible infrastructure, i.e., the Internet. Various Triple Space servers could be located at different machines all over the globe and hence every partner in a communication process can target its preferred space, as it is the case for Web and FTP servers. This highlights many advantages for providers and consumers. The providers 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.** There is a only minimal time dependency between the data provider and reader, in the sense that a triple must be written first before it can be retrieved.

- **Location autonomy.** The Triple Space as a storage location is independent of the storage space of the providers or readers 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 ex ante knowledge for purposes of communication through the Triple Space is not required. In the simplest case, the reading and writing of data is anonymous.

- **Data schema autonomy.** TSC provides its own schema (based on triples according to RDF) and the data written and retrieved from a Triple Space will follow that data model. This makes the provider and reader independent of their internal data schemas.

In addition it is worthwhile to state further positive side-effects of Triple Space computing:

- A Triple Space provides a trustworthy third-party infrastructure for data communication. Its involvement can enable secure data exchange and business processes negotiation and communication (see also Section 5 and Fig. 4).

- The Triple Space ensures persistent data storage. This guarantees that data is also available at a later point in time and that the data eventually can be read by all partners involved in the data exchange.

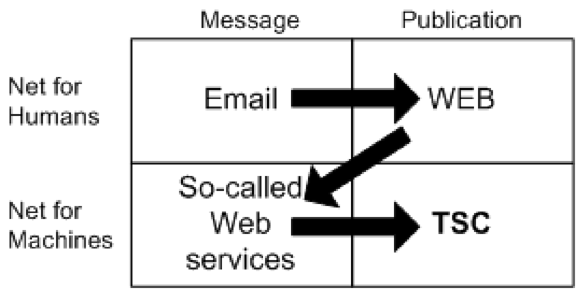

Figure 2. Evolution of communication means for humans and machines

In other words, Triple Spaces introduce an infrastructure that enables machines to use an equally powerful communication medium as the WWW provides for humans. Triple Spaces will supplement Web services, but will not replace current technological approaches. Just as WWW technology has advanced message-oriented communication means (e.g. phone or e-mail) for humans (Fig. 2), Triple Space computing provides a complementary approach for machine-to-machine interaction.

It is also clear that this is not the end but just the beginning of a long and promising endeavor for a revolutionary technology. No application can quickly check the entire Semantic Web whether there is a relevant triple. And vice versa, no application may want to simply publish a triple and then wait forever until another application is going to pick it up. Clever middleware is required that provides a virtual global Triple Space infrastructure without requesting each application either to download or to search through the entire Semantic Web. Moreover, the Triple Space needs to provide security and trust (see Section 5) without neglecting scalability. However, none of these requirements are really new. They apply for any application that works at a global scale.

In the next section we introduce a possible minimal architecture primarily based on existing Web technology. The architecture aims at providing a solution with as little functionality as necessary to implement the idea in a useful way.

4. A Triple Space architecture

A Triple Space is a virtual space that is identified by an URI. Triples are written and read with this URI as triple storage location. In that way a Triple Space must not be associated with a particular implementation nor with a particular storage system. This means that one implementation can host many spaces. For example, the two Triple Spaces *ts:deri.org/projectBudget* and *ts:deri.org/phoneNumbers* are virtual spaces as they store and manage 'their' triples. They might both be hosted by the same implementation *ts:deri.org/* or however by different servers. From a user perspective the location of a Triple Space is thus completely irrelevant.

The triples themselves are also denoted by unique identifiers (tID). There is however no required relationship between the tID of a triple and the URI of the Triple Space where it is located. In that way every triple is an individual object and can be addressed as such.

To write and read triples there exists a basic interface [5] that we abbreviate as follows:

- $write(triples)$
- $read(query)$

The read operation is non-destructive, i.e., a copy of the retrieved triples is returned (analogous to Web pages that are not erased when read). The returned triples are not replicas and changes in the Triple Space do thus not affect the retrieved copies. More details on the precise syntax of read and write operations are given in Section 4.2 in the part about the Triple Space Transfer Protocol (TSTP).

In the following section, we take a closer look at the components of our proposed Triple Space architecture and their interaction. We distinguish primarily three components — clients, servers, and spaces — and one interaction protocol. In Section 4.2 we discuss how the components could be implemented and how the TSTP protocol could be applied through a mapping to HTTP.

4.1. Components and component interaction

There exist three main components that interact in Triple Space computing (Fig.3):

- **TSC client.** A writer (publisher) is a TSC client and so is a reader (consumer). They may be distinct entities or in some cases the same. Clients are therefore not distinguishable from the viewpoint of a Triple Space, as every client can generally read and write triples independently.

- **Triple Space.** A Triple Space is a virtual entity implemented by Triple Space servers. The spaces within a Triple Space server are distinguishable so that write and read operations can be directed to the right space. In essence, a mapping from the read and write operation to the correct storage location has to be performed.

- **Triple Space server.** A server can host any number of Triple Spaces. TSC clients communicate with the Triple Spaces via the server by use of TSTP. TSC clients do however not know about the Triple Space servers, but only about the virtual Triple Spaces. Consequently, management operations have to be available on a Triple Space server to create (and possibly delete or empty) spaces [5]. The Triple Space server implementation can freely chose the storage facility that shall be installed. For example, one implementation might decide to implement storage on the file system and decides to put every triple in a separate file and every Triple Space in a different directory. Another implementation might prefer to have only a single file per Triple Space. Yet another one might decide to rely on a professional database system and to implement a Triple Space as a relational database.

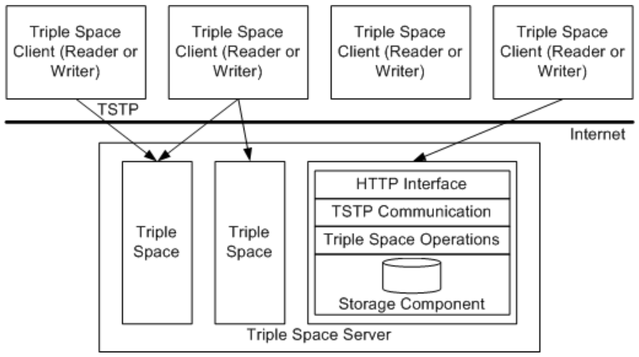
Triple Space Client (Reader or Writer)
Triple Space Client (Reader or Writer)
Triple Space Client (Reader or Writer)
Triple Space Client (Reader or Writer)
TSTP
Internet
Triple Space
Triple Space
HTTP Interface
TSTP Communication
Triple Space Operations
Storage Component
Triple Space Server

Figure 3. TSC System Architecture

As already mentioned, the TSC clients interact with the Triple Spaces by use of the Triple Space Transfer Protocol that provides five fundamental methods. A possible syntax for the write and read operations is

- tstp-write(tsURI,triple):tID
- tstp-write(tsURI,list<triple>):list<tID>
- tstp-read(tsURI,tID):triple
- tstp-read(tsURI,set<tID>):set<triple>
- tstp-read(tsURI,query):set<triple>

where every 'triple' is defined to be associated with an unique identifier (tID), as well as the three values subject, predicate, and object. In consequence triples are internally handled as quads of the form <subject,predicate,object,tID>. The 'tsURI' is the identifier of the Triple Space in question. Triples are either read by their identifier (tID) or by a 'query'. The queries are expressed in a RDF query language (e.g., SPARQL [19] or N3QL [3]). The syntax element 'tstp' denotes the protocol indicator. A possible way to implement the TSTP protocol is to define it in detail and expect all Web servers and application server vendors to adopt it. An easier approach is to map the protocol onto the HTTP protocol. This approach is proposed and presented in the next section.

4.2. Implementation

In this section we give an example of a minimal Triple Space implementation based on available technology and components.

TSC Client: As shortly indicated in the previous section it is possible to implement the TSTP on top of the HTTP protocol. This will allow any client that can invoke HTTP to also invoke the TSTP protocol. For convenience reasons it is however advantageous to have an implementation that exposes the specific read and write operation of the TSTP protocol. Thanks to this approach a software engineer would not have to worry about the TSTP-to-HTTP mapping and does not have to ensure a correct HTTP syntax.

Triple Space: The suggested Triple Space implementation is principally based on a layered architecture (Fig. 3):

- **Storage component.** The storage component is responsible for the persistent storage of triples. There are many alternatives, including file systems, relational or XML databases, RDF stores, persistent queues, etc.

- **Triple Space Operations / Application Layer.** Above the storage component is the core of the Triple Space. The Triple Space operations component implements the write and read operations for triples and ensures the appropriate error handling. Moreover it provides server management operations like creating, deleting, or emptying a Triple Space.

- **TSTP Communication / Presentation Layer.** This component interlinks the transport layer and the Triple Space operations layer: read and write operations are coordinated and while doing so, their consistency ensured. In particular this means that the TSTP communication component parses and serializes HTTP messages and checks the correctness of those.

- **HTTP Interface / Transport Layer.** As bottom layer of the Triple Space architecture, the HTTP communication component receives and sends the HTTP messages that wrap the TSTP protocol calls.

Triple Space Transfer Protocol (TSTP): The TSTP protocol is used by TSC clients in order to write or read triples. In this paper we suggest to layer the TSTP protocol on top of HTTP. In practical terms this means that TSTP directives are translated to HTTP. The translations are simple and the following mapping rules apply. The read method is shown for the use with a N3QL query and the write method for the use with a single triple. N3QL was chosen as it brings along several advantages compared to other languages: requests and answers use the same notation (N3), lighter syntax and thus less representation overhead as for example RDF/XML.

- **tstp-read** (ts:deri.org/phone,”<> ql:select...”) becomes

  ```
  GET /phone/?q=%3C%3E+ql%3Aselect...
    HTTP/1.1
  Host: deri.org
  Accept: application/rdf+n3
  ```

  and the response will be

  ```
  HTTP/1.1 200 OK
  Content-type: application/rdf+n3
  ...
  @prefix ex: <http://example.org/> .
  ex:prof ex:phone "6452" .
  ```

- **tstp-write** (ts:deri.org/phone,<ex:prof,ex:phone,”6452”>) becomes

  ```
  POST /phone HTTP/1.1
  Host: deri.org
  Content-type: application/rdf+n3
  ...
  @prefix ex: <http://example.org> .
  ex:prof ex:phone "6452" .
  ```

  and the response will be

  ```
  HTTP/1.1 200 OK
  Content-type: application/rdf+n3
  ...
  ```

This translation from TSTP to HTTP is simple and basically only relying on string rewriting. No additional information lookup and complicated re-serialization efforts are necessary.

In summary the short presentation of possible implementation aspects for a TSC architecture has shown that a minimal infrastructure based on well-known Web technologies is possible. We are currently working on a more comprehensive prototypical implementation.

5. Security and trust in Triple Spaces

One could argue that persistent, shared information spaces as an alternative communication paradigm create a wealth of issues with regard to security and trust. In this section we show that (1) trust issues in Triple Spaces are very similar to trust issues in communication based on message exchange and (2) argue that they can be handled effectively by reliably attaching the identity of the origin (the identity of the ”author”) to a triple rather than by mechanisms evaluating the facts stated by a triple. Additionally, we demonstrate that the combination of (a) reification and (b) disallowing change and delete operations on triples is a promising approach to reduce security and trust issues in Triple Spaces to a reasoning task.

5.1. From truth to trust in Triple Spaces

It is impossible to provide a mechanism that checks a triple for correctness of its entailed statement prior to adding the triple to the space. In other words, it is unavoidable that there will be triples containing ”false” statements. For example, anybody who has write access to a Triple Space could add a triple that says ”The world is flat”. However, this is not as fundamental a problem as it appears, since human communication in general, the Web, and message exchange-based communication have all worked well so far, notwithstanding the fact that people are making and publishing (intentionally and unintentionally) false statements. Surprisingly, we have no problem of retrieving ”true” facts from the Web despite the fact that it is easy for anybody around the world to publish false ones (and there are of course actually lots of false statements on the Web). This is because we have effective means of reasoning over the trustworthiness of a statement, and when doing so, often base this on the identity of the origin of the statement. When receiving an order from a customer stating ”We need 500 sheets of office paper”, it is not really important to know if the customer really needs this amount of paper (i.e., whether the stated fact is true), but rather whether we know the sender of the order and whether we trust him. Techniques like digital signatures based on asymmetric cryptography cover exactly this aspect: They ensure that the identity of the originator of a statement is reliably attached to the statement. The same approach can be easily applied to Triple Spaces: Anybody is allowed to add a statement, as long as it is made sure that the identity of the origin is kept in a reliable form, and that nobody can delete or forge existing triples.

5.2. Trustworthy Triples: Endorsing triples by authorship and reification

In the simplest case, the reading and writing of triples to the Triple Space is anonymous. While this might be acceptable for read access, it is problematic for write access, since this detaches a statement from the source of its origin and makes it impossible to take the origin into account when assessing the credibility of the statement. In brief, anonymous write access to Triple Spaces should be allowed only if we assume that statements from all sources are equally trustworthy. This can be the case if a Triple Space is used for integrating systems within one organizational entity, and if the only actors writing to the space are computer systems of the same business entity. In the following section, we outline a very simple yet efficient approach to use Triple Spaces by a wider, more heterogeneous group of people.

Our approach ”Trustworthy Triples” consists of two simple steps:

1. For each write access, we require authentication of the origin using standard methods (e.g. log-in based on username and password).

2. We translate each triple <my:sub,my:pred,my:obj> added during this session into a reified statement that is augmented (a) by a reference to the URI of its author and (b) the point in time the statement was issued. This would result in the following set of triples:

```
<ID, rdf:type, rdf:Statement>
<ID, rdf:subject, my:sub>
<ID, rdf:predicate, my:pred>
<ID, rdf:object, my:obj>
<ID, ex:addedBy,
http://www.deri.org>
<ID, ex:addedAt,
"2005-06-30-12:04:32">
```

ID in here is a unique identifier for this reified statement, to be created by the Triple Store. If, as proposed in our prototypical implementation, we use quads instead of triples for the internal representation (i.e., each triple has an identity and an identifier of its own), then the reification can be further simplified. ”addedBy” is a predicate pointing to the URI of the author of the original triple. The mapping between individuals and these URIs is established during enrolment of users to the Triple Store, and is to be managed by the Triple Store. ”addedAt” is a predicate pointing to a literal value that holds the point in time this statement was added. This is important for later reasoning about multiple contradicting statements issued at different points in time (often the latest statement will be the most trustworthy one in a chain of conflicting statements about the same property for the same resource).

This in combination with disallowing delete and change operations on the Triple Space provides exactly the functionality described in the subsection above: A multiplicity of users can add triples to the Triple Space, but nobody can destroy or maliciously modify existing statements.

When accessing the Triple Space, it is just necessary to restrict the triples to be returned to such added by a source we trust. Whom we trust can of course be expressed as a logical expression (”I believe all triples added by either Peter OR Mary” or ”I believe only triples added by Peter AND also added by Mary”) or in a full taxonomy, allowing very flexible access. The Triple Store can either return the

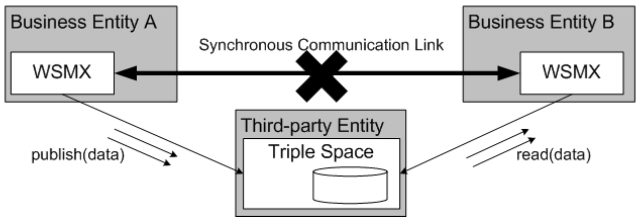

Figure 4. Usage of third-party Triple Space provider as envisioned by the TSC project

”raw”, reified statements, or transform those that match our trustworthiness conditions back into the original triples.

6. Related efforts

In the scope of the Triple Space Computing (TSC, http://tsc.deri.at) project, the authors aim at developing a middleware supporting coordination and communication with Triple Spaces (Fig. 4). The project’s main objectives are thus to extend Tuple Space computing to Triple Spaces and to make the Triple Space compatible with current Web service technology. In this way it is envisioned to overcome the deficiencies discussed in Section 2. The TSC technology will be integrated in the Web Service Execution Environment - WSMX (http://www.wsmx.org), which is the reference implementation of the Web Service Modeling Ontology framework (http://www.wsmo.org). Hence it is guaranteed that the project outcome is aligned with emerging technologies in the area of Semantic Web services.

Based on the definition of the Triple Space architecture we want to further improve the implementation. A major issue is the underlying storage component. ’Yet Another RDF Store’ (YARS, [13]) is of particular interest, as

- it provides a data store for RDF in Java,
- it uses N3 to encode facts and queries, and
- it already handles the concept of quads.

The interface for interacting with YARS is plain HTTP (GET, PUT, and DELETE) and is built upon the REST principle. YARS supports keyword-based searches and content negotiation. Therefore the efforts of the YARS research group are of great interest to our work with TSC.

In addition to the efforts of the TSC project consortium there exist other initiatives [15][20] that want to bring semantics to Tuple Space computing. [15] introduced, in the scope of the sTuples project, a Tuple Space supporting semantic data that extends the JavaSpace [11] implementation. A generic Semantic Tuple containing an object field of type DAML-OIL Individual extends the JavaSpace Entry interface. Fundamentally, a non-semantic implementation of Tuple Spaces was augmented with Semantic Web technology. There are however various difference to the TSC approach: sTuples targets in-house infrastructures for e.g. intelligent conference rooms. Moreover, the underlying data model is a mix of non-semantic and semantic technology (compared to clean semantic triples/quads in Triple Spaces). All in all, sTuples point in the same direction, but do not aim at an Internet-wide system that is fully supporting Semantic Web technology in the Web sense. In [20] the authors introduce RDFSpaces, which continue their previous work on XMLSpaces. The latter is a Java and .NET implementation of the classical Linda model that was extended in order to support the management of XML documents as types of tuples and tuple fields.

7. Conclusion

Ubiquitous computing is expected to be the ”killer application” for Tuple Space computing [14] because of its portability, extensibility, flexibility, and ability to deal with heterogeneous environments. In fact, we think that space-based communication is close in spirit to the Web and may help to bring Web services to their full potential. It requires a move from a message-oriented communication model to a Web where information is persistently published and read. We have reason to believe that Tuple Spaces help to overcome many problems around heterogeneity in information distribution and information access. Since applications are decoupled in reference, time, and space, many issues in protocol and process alignment are mitigated. However, Tuple Space do not contribute to the solution of data and information heterogeneity. In this paper, we proposed to combine the idea of Tuple Space computing, based on shared information spaces, with Semantic Web technology. Our approach is to augment Tuple Space into a semantically-enriched Triple Spaces holding RDF triples. This Triple Space adds to the Web a means for the exchange of semantic data between applications and services. Therefore, the Triple Space may become the Web for machines as the Web based on HTML became the World Wide Web for humans.

Acknowledgment

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT projects RW^2 and TSC; and by NCR Korea.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*, Springer, 2003.

[2] T. Berners-Lee, J. Hendler and O. Lassila, ”The Semantic Web”, *The Scientific American*, May 2001.

[3] T. Berners-Lee, ”N3QL - RDF Data Query Language”, July 2004,
http://www.w3.org/DesignIssues/N3QL.html

[4] D. Brickley and R.V. Guha (eds.), ”RDF Vocabulary Description Language 1.0: RDF Schema”, W3C Recommendation, Feb 2004,
http://www.w3c.org/TR/rdf-schema/

[5] Ch. Bussler, ”A Minimal Triple Space Computing Architecture”, 2nd WSMO Implementation Workshop, Innsbruck, Austria, June 2005.

[6] 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

[7] D. Fensel and C. Bussler, ”The Web Service Modeling Framework WSMF”, *Electronic Commerce Research and Applications*, 1(2), 2002.

[8] D. Fensel, ”Triple-space computing: Semantic Web Services based on persistent publication of information”, Proc. of IFIP Int’l Conf. on Intelligence in Communication Systems 2004, Bangkok, Thailand, Nov 2004:43-53.

[9] R.T. Fielding, ”Architectural styles and the design of network-based software architectures”, PhD Thesis, University of California, Irvine, 2000.

[10] R.T. Fielding and R. N. Taylor, ”Principled Design of the Modern Web Architecture”, ACM Transactions on Internet Technology (TOIT), 2(2), May 2002:115-150.

[11] E. Freeman, S. Hupfer and K. Arnold, ”JavaSpaces Principles, Patterns and Practice”, *The Jini Technology Series*, Addison-Wesley, Reading MA, USA.

[12] D. Gelernter, ”Generative Communication in Linda”, ACM Transactions on Prog. Lang. and Systems, 7(1):80-112.

[13] A. Harth and St. Decker, ”Yet Another RDF Store: Perfect Index Structures for Storing Semantic Web Data With Contexts”, June 2004,
http://sw.deri.org/2004/06/yars/doc/summary

[14] B. Johanson and A. Fox, ”Extending Tuplespaces for Coordination in Interactive Workspaces”, *Journal of Systems and Software*, 69(3), Jan 2004:243-266.

[15] D. Khushraj, O. Lassila and T. Finin, ”sTuples: Semantic Tuple Spaces”, 1st Ann. Int’l Conf. on Mobile and Ubiquitous Systems: Networking and Services, Boston, USA, Aug 2004:268-277.

[16] G. Klyne and J.J. Carroll (eds.), ”Resource Description Framework (RDF): Concepts and Abstract Syntax”, W3C Recommendation, Feb 2004,
http://www.w3.org/TR/rdf-concepts/

[17] D.L. McGuinness and F. van Harmelen (eds.), ”OWL Web Ontology Language: Overview”, W3C Recommendation, Feb 2004,
http://www.w3c.org/TR/owl-features/

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

[19] E. Purd’hommeaux and A. Seaborne (eds.), ”SPARQL Query Language for RDF”, W3C Working Draft, Apr 2005,
http://www.w3.org/TR/rdf-sparql-query/

[20] R. Tolksdorf, L. Nixon, E. Paslaru Bontas, D.M. Nguyen and F. Liebsch, ”Enabling real world Semantic Web applications through a coordination middleware”, 2nd European Semantic Web Conf., Heraklion, Crete, June 2005:679-693.

[21] M. zur Muehlen, J. V. Nickerson and K. D. Swenson, ”Developing Web Services Choreography Standards - The Case of REST vs. SOAP”, *Decision Support Systems*, 37, 2004.