Queues Are Spaces - Yet Still Both Are Not The Same?

D. Fensel[1]
E. Kühn[2]
F. Leymann[3]
R. Tolksdorf[4]

13 May 2007

[1] DERI Innsbruck, University of Innsbruck, Austria
[2] Institut für Computersprachen, Space Based Computing Group, Vienna University of Technology, Austria
[3] Institut für Architektur von Anwendungssystemen, University of Stuttgart, Germany
[4] Institut für Informatik, Free University of Berlin, Germany

Abstract. This paper compares two different paradigms for machine-to-machine communication: messaging and space-based communication. As with human communication, one can distinguish between message-based, directed communication (*email* – abstracted into “messaging”) on the one hand, and anonymous communication based on persistent publication (*Web* – abstracted into “spaces”) on the other. However, despite these obvious differences, a closer analysis reveals that messaging can simulate space-based communication, as well as the other way around, and that both communication styles can, in a certain sense, be viewed as an implementation layer for the other. In our paper, we elaborate on these contradicting views and draw conclusions on the relationship between messaging and spaces on conceptual, technical, functional, and pragmatic levels. Conceptually, space-based communication will replace messaging in heterogeneous environments. Technically, messaging will implement many space-based communication solutions. Functionally, both technologies will nearly perfectly simulate one other. Pragmatically, it must be acknowledged that while, in principle, space-based communication may indeed be the “better” technology, in comparison to the mature message-based industry standards, it faces significant challenges at becoming an adequate and adopted machine-to-machine communication mechanism. However, as the Web established itself as a complementary communication technology five years after the widespread adoption and maturation of email, this point of view may be simply too pessimistic. As usual, it is difficult to predict the future – even, or especially, for the experts in the field.

Table of Contents

1. Introduction

There is a lot of interest in combining Semantic Web technology [Berners-Lee et al., 2002] with Web service technology [Weerawarana et al., 2005]) in order to make the discovery or even the composition of Web services much more facile and precise [Cardoso & Sheth, 2006, Hendler et al., 2002]. By adding semantics to Web services, a service requestor may specify the service needed in terms of its problem domain, i.e. using business terminology. This is a major progress compared to the current state of the art in the field. To date the discovery of Web services is mostly based on signature matching, on the QName of an implementation of a port type needed, or simply just on pre-defined addresses of the port to use [Weerawarana et al., 2005].

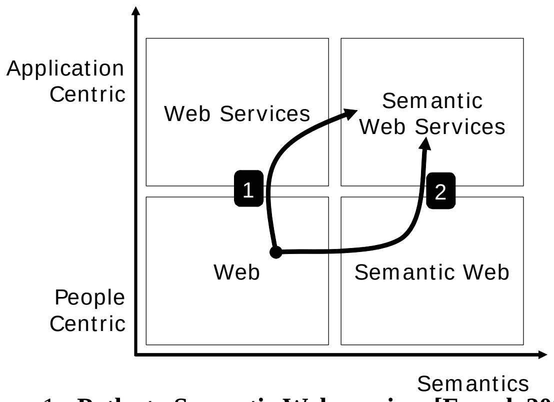

Figure 1 - **Paths to Semantic Web services [Fensel, 2004]**

Figure 1 depicts the phases in which the **Web** evolves [Fensel, 2004]: The pure Web is centered on supporting human beings sharing information; however this information is not semantically annotated. The **Semantic Web** adds semantics to Web content, describing Web resources in terms of the business domain they provide information about, in order to support human beings in discovering the most appropriate information. **Web service** technology is about exposing (application) functionality at network endpoints which are reachable over various transports. Web services are intended to be used by applications and not by human beings directly. As a consequence, semantic descriptions for Web services have not been considered when this technology initially appeared. **Semantic Web services** are Web services enriched by semantic descriptions for the purpose of enabling improved service discovery mechanisms. The technology supporting **Semantic Web services** can be achieved in one of two ways (ref. Figure 1):

- First, Web service technology can be extended with semantics, i.e. it can be combined with corresponding technology developed in the Semantic Web field in the last decade.

- Second, Semantic Web technology can be extended by middleware technology, i.e. combining successful or promising middleware technology with Semantic Web technology to enable service-oriented computing. Triple-based computing [Riemer et. al, 2006] as envisioned in [Fensel, 2004] is about the second path: it strives towards applying the concept of "spaces" [Wyckoff et al., 1998] to the Semantic Web [Khushraj et al., 2004]. For this reason this combination is also commonly referred to as "triple-space computing" [Khushraj et al., 2004].

Space computing has its origin in parallel programming ([Gelernter et al, 1985]). Here, a "space" is a place where data can be shared by multiple components. A component can easily put a piece of data to be shared with others into a space ("out"). Any number of components can read a piece of data without removing it from the space ("rd"); a component may read the same piece of data multiple times. There is also a variant of reading data that removes data from the space once it is read (destructive read, referred to as "in"). A space may store data persistently such that it is not lost when the environment realizes an outage. The data model supported by a space is originally that of a "tuple", which is why a corresponding space is also referred to as "tuple space". In a tuple space, data to be read (or taken) is identified by a template, i.e. by specifying values in certain slots of a tuple to be retrieved while other slots are left open. If more than one tuple will match a template, only the first match will be returned as a result. To support more sophisticated applications, advanced features for manipulating tuples in a space are defined (cf., for example, [Kühn, 1994, Kühn, 2001] for the description of a virtual shared memory model with unique object references, distributed flex transactions, and configurable replication strategies, [Lehmann et al., 1999] for implementation considerations). Finally, matching of tuples to be retrieved from a space can be supported by associating semantics with each tuple (so-called "sTuples", [Khushraj et al., 2004]). There are two major similarities between the architectural principles of the Web and tuple spaces:

- As observed in [Fensel, 2004], the variant of a persistent space where data is reliably shared by means of unique identifiers in a non-destructive manner is exactly how the Web works [Fielding, 2000]. Identifying Web resources by means of URIs is considered as one of the underpinnings of the architectural style behind the Web (the so-called REST style, [Fielding, 2000]) and it is the major ingredient for Web scalability because it enables caching of resources. Thus, it is an obvious conclusion that in order to allow scaling up spaces to the Web size, tuples have to be identified by URIs, too.
- Another underpinning of the architectural style of the Web is that of a very simple set of methods for accessing resources. A core of four methods (the so-called CRUD methods) is provided for creating, reading, updating, and deleting a resource [Fielding, 2000]. These four methods are the key methods supported by HTTP. In [Bussler, 2005] it is argued that it is only natural to consider a protocol similar to HTTP as the

underpinning of communication in a Web-scale triple space computing environment.

Bringing together the (Semantic) Web and space-based computing leads to a triple space environment (cf. Figure 2). A triple space is a logical concept, a container for triples. Triples consist of tuples with associated semantic descriptions. Triples are manipulated in the context of a space. A space itself is identified by a unique identifier (e.g. a URI, as suggested in [Bussler, 2005] and [Kühn et al. 2005a]). The overall environment consists of multiple spaces (like the Web is made up of multiple domains). The persistent property of a space is realized by underlying storage mechanisms like file systems or database systems where the triples are stored. These storage mechanisms may be provided by multiple servers, each of which stores a fragment of a subset of all spaces (just as Web servers store the resources manipulated via HTTP requests).

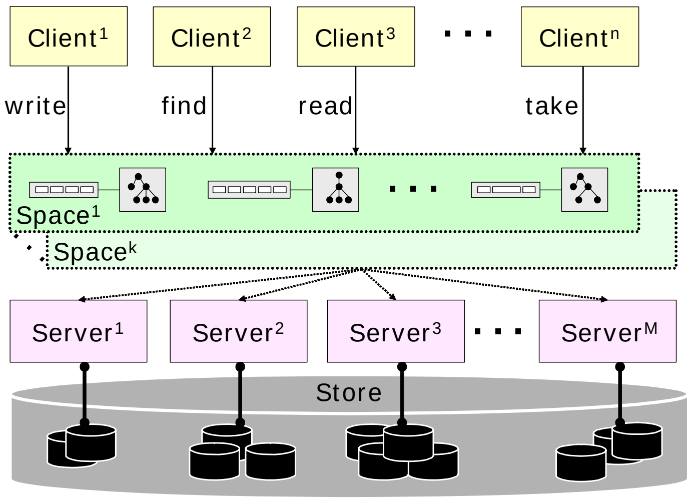

Figure 2 – **Triple Space Environment**

Merging space-based and Web service technology would bring Web services closer to the major principle of the current Web which is based on persistent publication and not on messaging as it is the case for current Web service technology. Besides their name, Web services do not have much to do with the Web. Let’s illustrate this briefly by assuming a time machine would bring us back to the pre-Web time. What was a very common way, back then, of accessing a research paper? One was posting an email kindly asking for the paper and a friendly colleague was replying this email, including the requested document as an attachment. Dissemination of information was based on message exchange. The communication overhead in publishing and accessing information was high and dissemination was therefore quite limited and slow. With the rise of the Web the situation changed significantly. The author could publish the paper once by making it accessible on his Web page, and then focus on writing new papers, rather than catering to the requests of other researchers. New services such as citeseer[1] even ensure durability of this publication beyond the life time of a Web page - they disable the delete operation of the space of publications. All the potential readers could get instant access to the paper without requiring a two-stage message-exchange process. This tremendously scaled and sped up the information dissemination process. When comparing Web services with this essential Web principle it becomes quite obvious that Web services are **not** about the Web. It is true that Web services can use the Internet as a transport medium, relying on protocols such as FTP, SMTP, or HTTP, cf. Section 3.1. However that is all they have in common with the Web [Fielding, 2000, Fielding & Taylor, 2002, zur Muehlen et al., 2004]. When sending and receiving SOAP messages, the content of the information is hidden in the body of the message and is not addressed as an explicit Web resource with its own URI. Therefore, all Web machinery involving caching or security checks is disabled since its use would require the parsing and understanding of all possible XML dialects that can be used to write a SOAP message. Referring to the content via an explicit URI in an HTTP request would allow the content of a message to be treated like any other Web resource.

The major objective of this paper is to better understand the potential of space-based approaches for Semantic Web services. We will show that the differences between messaging and space-based communication are not as blatantly obvious as what may be concluded from an initial analysis. In fact, it appears that messaging can simulate space-based communication, and alternatively, space-based communication can simulate messaging as well. Moreover, both communication styles can, in a certain sense, be viewed as an implementation layer for the other. In our paper, we elaborate on these contradicting views and draw conclusions on the relationship between messaging and spaces on conceptual, technical, functional, and pragmatic levels.

The structure of the paper is as follows. In Section 2, we analyze fundamental features of communication and how these features appear in communication enabled through the usage of the Internet. Sections 3 and 4 introduce messaging and space-based computing, respectively, as communication paradigms for Web services. In Section 5, we discuss how the each paradigm can be used to simulate the behavior of the other and that nearly any requested functionality can be (at least in principle) provided by both to a similar extent. This actually recalls an insight of [Gray, 1995] that elaborated the relationship between queues and databases, pointing out that **"Queues are Databases"**. Section 6 attempts to unify the contradicting results of our analysis - having two different communication paradigms that, contrarily, seem to be the same in many respects - and to draw expectations for the further development path of Semantic Web services.

[1]http://citeseer.ist.psu.edu/cs

2. Communication Through Space and Time

The purpose of this section is to discuss some basic principles of communication that underlay the paradigms addressed in the following sections. After having discussed these principles we briefly sketch three major communication styles enabled through the usage of the Internet.

In general, communication is a means to synchronize the activities of living beings. It is achieved through channels in their physical environment which become a transport layer for meaning. This alternation must be durable for an adequate period of time to allow the desired receiver to become aware of it. In a sense, it should be as permanent as possible to achieve this purpose; however, this is not really practical. Reasons that make a limited duration acceptable are:

- The information is of temporal nature.
- The receiver is known and we assume that he/she can store the information through a modification of his/her internal physical existence, i.e., he/she internalizes the physical modification.

In contradiction, this also indicates the rationales why a longer duration for the representation of information might be required:

- The information has a long-term value.
- The receiver is unknown, i.e., anonymous.
- The receiver tends to forget things.

Human communication was first based on verbal and non-verbal gestures. Resultantly, only the transmission of temporary information for a small number of receivers was possible. Most of the information was immediately lost. As soon as our social organisms established more structure and size, the transmission of information beyond the immediate context of the speech act became a necessity. Concomitantly the society also required a means to store information beyond the memory of its individual members. Stones started to carry meaning and became later replaced by papyrus, paper, and printing technology. Through these storage media transmission of information was clearly separated from the single communication act and the memory of the detached individuals. Anonymous communication beyond the immediate boundaries of space and time became possible.

However, the boundaries remain fragile. On the one hand, a letter can be vocally transmitted or sent as a printed letter, which then runs the risk of being read or received by those other than the indented receivers; this information can even be published, as the letter turns from a targeted message into a broadcasted publication for anonymous receivers. On the other hand, think about a printed PhD thesis as one of the most famous examples for write-only documents.

In summary, persistence and anonymity are related in a complementary manner and their difference is not in principle, but rather pragmatic. Instead of publishing a book we can write a letter to many unknown people (aka spam), and vice versa, many letters, originally intended for single individuals or groups, end up being widely published.

Does the Internet change any of these? First of all, since the Internet is just a technical means of communication, the insights addressed above do not change. Rather the Internet requires a refinement of these principles within the given properties of this specific medium of communication. Still, each medium of communication comes with a specific impact on the way communication is shaped by it. The Internet has around forty years of history and was initially established as a protocol for inter-machine communication. The important design features were to bridge heterogeneous operating systems and to minimize the risk of failure of central nodes in the network.

Around 25 years after its invention, the Internet had its first "killer app" that significantly shaped modern communication: **email**. Instead of exchanging paper letters, people realized the advantage of sending letters electronically. It had certain disadvantages - especially in the time where no attachments, or MIME standard were around - but it was simple, fast, effective, and cheap. Thus, the first major application of the Internet was **message-based communication of humans**.

Around five years later, a second killer app of the Internet appeared: **human communication based on persistent publication**, the **Web**. With the Web, people no longer needed to send the information to each potential receiver and require him/her to store the message. Instead, a piece of information was now persistently published - as long as it is not changed by the owner of the information - and anonymously accessible. It was no longer needed to anticipate the potential receiver of the information. Now, with more than 20 billion Web pages and more than 1 billion of Web users, this publication-based communication medium has revolutionized human communication. The two communication styles have a certain degree of overlap in terms of, for example, **email lists**, used to broadcast information, and **archived email lists**, used for persistently storing email communication. However, such email lists do not provide a means to uniformly access the persistently stored information, which is a central design principle of the Web and, in the same time, one of its key success factors.[2]

[2] This is the same in nature. As soon as we have identified the difference of animals living at the land or in the sea, we are immediately confronted with species that blur this distinction. This, however, is necessary because otherwise the transition from sea-based to land-based animals would need to be explained through a miracle. Actually, the existence of these transition species is a necessary step in explaining natural evolution.

Another five years later, a new fundamental communication paradigm provided via the Internet became popular: **Web services**. Just as email communication - and in opposition to the Web paradigm - Web service technology is based on message exchange (e.g. SOAP messages). However, contrarily to email, Web services are not explicitly targeted at facilitating human interaction, but are rather a means for automated application integration, i.e., for mechanizing communication between programs that provide certain services to a human user based on their interaction. Through Web service technology, the Internet has become a major means for **machine-to-machine communication (based on messaging).**

Drawing an analogy to the evolution of the human communication network addressed above, a natural question arises: is messaging the only paradigm for proper machine-to-machine communication or will we see something evolving such as the **Web for machines**, i.e., a communication means based on persistent publication for anonymous consumption of information rather than tight coupling of information exchange through defined receivers and senders of information? Should the Internet evolve in a similar fashion as the human communication network, as our analogy suggests, then the latter progression is not at all unrealistic. Further exploring this question is the major purpose of the remainder of this paper. An evolution towards the Web of machines would trigger monumental changes, as the number of machines available is several orders of magnitude higher than the human population, whilst the information processing capability of the former in regard to symbolic information is incredible fast and exponentially growing.

3. Messaging

Web service technology has been invented to solve a couple of challenging application integration problems [Weerawarana et al., 2005]. The focal points of application integration in this paper are the messages to be exchanged between applications. Message exchange in its simplest sense requires transporting messages from the sender to the receiver.

Bindings and transports - Within enterprises, transport mechanisms such as JMS or RMI/IIOP are of much higher importance with regard to application integration than HTTP. Due to this factor, Web service technology has been architected from the outset to support any transport mechanism. The vehicle by which different transport mechanisms are supported is via the construct of a binding (e.g. WSDL). In a nutshell, a binding specifies how a message is sent from a source to a target (the service), in other words which transport protocol to use, how to serialize a message as payload in the transport envelope, and how an endpoint reachable via a certain transport protocol (so-called port) is identified in a transport protocol dependent manner. As the concept of a binding reveals, taking the prefix “Web” in “Web service” too serious may be misleading, provoking statements about Web service technology that are incorrect. The critique that is absolutely valid to make is that using the prefix “Web” was not a good idea; just “service” would have been sufficient and more correct from a technical point of view. Web services can of course be used over Web transports using Web technology (via appropriate bindings), but Web services have been explicitly invented to be used in non-Web environments too. In fact, many important applications of Web service technologies make no use of Web technology at all - especially they make no use of HTTP. For instance transmitting Java objects over JMS is an often used binding within Web service applications, cf. [WSIF] for more details about this and other bindings.

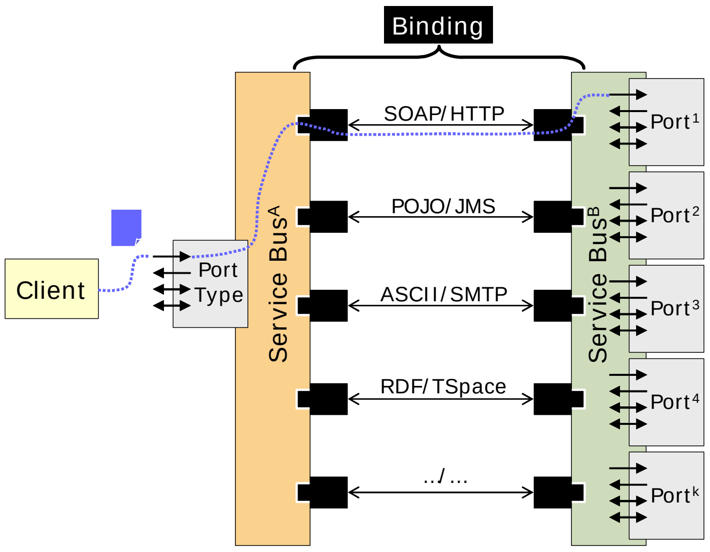

Figure 3 - Bindings Enabling Communication via Multiple Transports

3.1 Bindings and Spaces

The before-mentioned JMS binding has interesting properties when compared with an HTTP binding:

- The JMS binding transports the message reliably, i.e. it will guarantee the delivery of the message by making it persistent along its path to the recipient.
- The JMS binding supports asynchronous communication by delivering the message into a queue from which the recipient can get it and process it at a time convenient for him.
- The JMS binding could support publishing a message on a topic, i.e. multiple recipients could get (copies of) the message.
- The JMS binding supports that sender and receiver of the message do not have to know each other at all.

These are properties of a message exchange that originally motivated the use of spaces for communication with Web services [Fensel, 2004]. As discussed above, spaces provide features like repetitive reads of the same message by one and the same recipient that are not supported by the JMS binding sketched. Consequently, two threads of investigations are worthwhile in order to further investigate the relationship between message- and space-based communication:

- Define the Web service scenarios that require repetitive receipt of the same message by a given Web service

- Define a “space binding” for Web services, i.e. a binding that uses space technology for message exchange in Web service communication such as the JMS binding uses queue technology or pubsub technology for the same purpose

3.2 The Service Bus and Multi-Protocol Services

For a requesting client only the interface of the service (so-called port type) is of interest. The client passes the message targeted to a service to its supporting middleware (so-called service bus) indicating the service interface needed. The service bus then determines (based on a variety of criteria) a service implementing the required interface, and an appropriate binding to be used to actually carry the message over to the receiving service, in other words the service bus performs service discovery on behalf of the requesting client. Typically, a binding corresponds to a handler, a plug-in of the service bus understanding how to use the particular transport protocol, etc. (cf. Figure 3). Note that it is absolutely valid and even intended that one and the same service is available at multiple ports over multiple protocols (so-called “multi-protocol services”). The decision how services and bindings are selected by the service bus is highly non-trivial and is influenced by a broad spectrum of parameters such as policies, service level agreements, workload considerations ([Leymann, 2005]). In general, these patterns are not specific to message exchange but have been developed in this context.

3.3 Message Exchange Patterns

An operation of a Web service always follows a message exchange pattern [Weerawarana et al., 2005]. A message exchange pattern (MEP) specifies the order in which an endpoint providing the functionality of the operation expects certain kinds of messages or emits certain kinds of message. A straightforward example for an MEP is a request-response operation that, first, expects the request message and, second, returns the response message. A more complex MEP is a request-for-bid operation that first sends out a message, then expects multiple messages of another type back. Note that a MEP often has a matching “dual MEP.” For example, the MEP dual to the request-response MEP is called “solicit-response” MEP. This MEP first sends out a message that contains the request to be executed by a receiving service, and it expects a message back with the response resulting from the execution of the service. It is important to note that the concept of an MEP and the concept of a binding are orthogonal, whilst of course proper bindings ease the implementation of a MEP. An implementation of the request-for-bid MEP benefits from a binding that supports multicast of messages like a JMS pubsub binding or like a space binding envisioned above. Thus, it is a worthwhile undertaking to develop basic MEPs that significantly benefit from space bindings when compared with other bindings. Vice versa, it is a worthwhile investigation to define which usage patterns of spaces can be described by appropriate (pairs of dual) message exchange patterns.

4. Space-Based Communication

Space-based computing dates back on the invention of the **Linda** model by David Gelernter in the late eighties. The original Linda model was intended for parallel computing and later on also used in the context of distributed computing. A lot of research work has been motivated by the Linda model; we have seen many bindings to different programming languages (C, FORTRAN, and Prolog etc.) and connections to domain specific languages like XML, extensions for multiple spaces, access rights and transactions etc. By the mid-nineties, the Linda model experienced a new revival with the language binding to Java, called Java Spaces [Freeman et al, 1999], as part of the Jini standard [Waldo, 1999].

Besides Linda, virtual shared (or distributed shared) memory can be seen as another research track related to space-based computing. The differences concern how synchronization is programmed, reaching from low-level locking to high-level transactions found in models such as Linda, and the structure and granularity of the data, reaching from sharing entire memory pages to single objects or structured records again found in models such as Linda.

Common to both Linda and virtual (distributed) shared memory is the notion of a common, abstract data space connecting distributed processing entities over a network. This leads to a new understanding of how communication is carried out which can be illustrated by the following two examples:

- Consider the management of an intersection. In the first solution, each car is equipped with a mobile phone and must directly call – send a message – to each other car on the road in order to avoid crashing. The second solution implements a traffic light as a commonly shared device that can be observed by each party consistently. The latter, albeit a simplified scenario, refers to the usage of a shared space. Its abstraction makes optimizations possible like changing policies for the phasing. An extension would be that the traffic light can observe the ongoing traffic and perform dynamic decisions.

- Another example would be a teacher using a blackboard to cooperate with all students concurrently, instead of speaking directly with each single student. Besides that this is more efficient, students (cf. mobile devices) that join the room later, or decide to not listen for a while, can immediately catch up at any time in order to see what is going on. The traffic light as well as the blackboard represents stateful communication. The tracking and monitoring of what has been said is easy since the messages are kept in the space[3] and are not only observed by receivers

[3] Pre-messaging EAI based on file sharing had the big problem of garbage collection, i.e. how to detect that all interested parties read the file, and who is going to delete it. The issue of garbage collection in space based systems has been addressed with first solutions like [Menezes & Wood,

available at the time the message is sent; in other words, messages in a space are not "lost" after consumption. The advantages of such a system are clear: the slowest student must no longer complain if the teacher erases the board too early; multiple reading of the same message is possible. For instance students that did not understand the formula on the board can re-read it at a later time.

Both examples demonstrate that a space is a more natural and efficient way for communication than explicit message sending. The space offers an abstraction to pure message sending. The latter reflects physical communication lines. Comparing this evolution to higher level programming languages that have been developed based upon the "von Neumann" hardware-related style of programming, an advantage is that programming becomes easier and that with higher level software abstractions new application scenarios are enabled. A further characteristic of space-based computing is that participants can be automatically notified if data upon which they are waiting arrives in the space. The association in Linda is done via template matching - this can be compared with query by example. Thus queries can be issued before the desired data is there, referring to the decoupling concerning time, space and reference [Kühn et al., 2005b].

Conceptually, a space can be seen as a database that serves for communication. Linda tuples refer to records in a non-normalized database table without primary keys, the "out" operation relates to insert, the "read" operation to a select statement, and the "in" operation to a read followed by a delete statement in the same transaction. Triggers can be used to implement notification. Indeed, many Linda implementations use a database underneath. Distributed applications are built by allowing an application to access several spaces (databases):

The idea of considering space-based communication for the Semantic Web has led to the concept of Triple Space [Fensel 2004, Riemer et. al., 2006] maintaining semantic information in a structured and reliable way. Triple Space emphases the use of the space for the publication and retrieval of "payload" data that represent the know-how published on the Web. The structured representation as RDF triples in a space allows for reasoning about this semantic data. The space guarantees the reliable and efficient placement of this knowledge, and additionally provides a smart interface including near-time notification about changes.

1999]. However, the fact of exponential growth of information makes deletion of old content less of a relevant problem anyway.

5. Is There Any Difference?

In Sections 3 and 4 we discussed messaging and spaces as distinct and independent means of communication. The purpose of this section is to blur this distinction. We will show how each of the different means can just be viewed as a special case of its opponent.

The discussion whether one can implement a space with a queue must be rendered more precisely. What does “A can be implemented with B” mean? Clearly a space must be implemented by means of sending a message since the physical common memory does not exist. This is the reason why more sophisticated communication and replication protocols that are peer-to-peer based will be needed in the future to maintain distributed memory spaces [Kühn, 1994]. So let us define that “A can be implemented with B” does not imply that A and B are conceptually equal. Rather the question is whether A can provide a useful abstraction over B in order to be allowed to call itself a new stand-alone concept or paradigm. Obviously we would see this to be the case for queues. The justification lies therein that queues have proven to be the most useful communication pattern we have experienced so far and thereby indeed deserve particular optimizations.

Space-based computing at a low level of consideration such as Linda operations probably can not compete with an advanced pattern like a queue, albeit offering attractive basic coordination ideas like notification and the “in” operation. So we will rather consider patterns implemented with space-based computing and relate these to queue based patterns. A few interaction patterns and scenarios that are native to space-based computing and require a more complex implementation with queues are:

- **Multi-readers pattern -** A consumer observer scenario such as a chat room where many producers send new messages and where all consumers may (re-) read the produced messages any time and in any order.
- **Request/answer pattern -** Assume that there are many requestors that dynamically join and leave and that servers need to distribute the load between them fairly; each server could subscribe to several spaces, pick up work according to its capacities, and write the answers back tagged with the corresponding sender’s identifications. This pattern is useful to cope with peak-times. If the load becomes too high, further server sites could be activated to take over work dynamically without the requestors being aware of it.
- **Extended queue pattern** - Queues in which the sender might change or remove an already queued message or in which the receiver might apply another consumption order other than the FIFO order (for example random order which is quite natural to implement with Linda spaces, subject based

order which relates to topic queues such as pubsub, and could also be enhanced through random access or any other priority related access).

- **Market place patterns -** Patterns involving tighter bi-directional n:m collaboration like negotiation, bidding, voting etc.

5.1. How a Space Can Simulate Messaging

Space-based communication abolishes direction of communication. An “out” operation does not target a recipient, i.e. it does not determine an endpoint that has to read or consume the emitted tuple. For the simulation of messaging with spaces we must implement both, an order between the messages that are represented by tuples in the space, and a connection between senders and receivers of messages - in space-based terminology, producers and consumers of shared data.

If we think of the classic Linda model which does not provide further structuring, both abovementioned issues must be implemented explicitly. This can be achieved by adding additional fields to the tuples. One parameter could be added to hold the name of the receiver - if we want to simulate queues - or the name of a topic – for a behavior similar to topic queues and pubsub systems. A further parameter could be added to contain a counter that represents the order in which tuples are published and the order, in which they shall be read, be that in a destructive or a non-destructive way. In doing so we can already notice an extension to messaging, as the consumer is free to decide whether to read tuples destructively or non-destructively, to re-read them, or to completely bypass this order. We can encode additional information to establish further patterns of interaction into the general tuple-format. For RPCs this has been done in [Tolksdorf, 1998] where services are described by their interfaces.

For space conceptions that foresee more structure, the amount of self-programming when simulating messaging can be reduced. [Agha & Callsen, 1993] have explored such a concept with their ActorSpaces. Here, the classic actor model is coupled with a Linda-like communication style. ActorSpaces use patterns of tags as regular expressions to identify groups of receivers for a tagged tuple, e.g. group-size 1 is declared an individual receiver. A message is associated with a pattern which identifies receivers. Two primitives allow addressing of the message to one “out” of a group of receivers identified by the pattern (send) or to all of them (broadcast). Receivers are generated as “actors” and are registered under a name when entering the system. That name is compared to the receiver identification of a message-tuple sent. In an open system, however, broadcasts are technically not feasible and also organizationally not desirable. ActorSpaces therefore use the well-known approach [Gelernter, 1989] to use subspaces in order to structure a global tuple space. In [Kühn et al. 2005a] a generalization of tuple spaces is described where containers represent structured spaces that offer already order principles like

FIFO, and can be named with an URL. This URL can serve as the topic to which consumers may subscribe.

At the technical level, the architecture of a distributed space [Kühn et. al, 2005a] employs both: databases, for the reliable, efficient, and persistent management of tuples, and queues, for the reliable communication between distributed spaces. In other words, a space can be seen as an abstraction layer for message-based communication, and a queue can be seen as a special form of a space supporting FIFO-based communication. Hence if technologies such as established message queue products are used to implement spaces, the dispute about the maturity of space-based computing can be slightly pacified.

A major advantage of space-based communication is the generality of the initial model paired with the simplicity of operations. Since behavior will be restricted by extensions to the original model, the simplicity can be maintained if extensions are well-designed. The major drawback is of course the poor functionality of the basic model. In the Linda literature, there is a huge body of proposals for extensions that each solve problems such as the lack of fault-tolerance, transaction support, or real-time capabilities. However, it is usually not well understood whether and how these extension interact and depend on each other. A final drawback is the scalability of tuple space systems, i.e., there is currently very little work focused on scaling Linda to Web-size.

5.2. How Messaging Can Simulate a Space

One application area envisioned to evolve to a level of fundamental importance for Semantic Web technology is application integration [Hendler et al., 2002]. Figure 4 shows a traditional set up very often found in today's enterprises; this set up is typical for solving enterprise application integration problems (see, e.g., [Hohpe & Woolf, 2004]). The predominant technological underpinning of integration environments is reliable messaging.

Reliable Messaging Environment Architecture - A reliable messaging environment consists of multiple channels, where a channel is a logical place to share data. Data on a channel is referred to as a message (instead of a tuple). An environment may have many different channels. And each of these channels may be supported by multiple servers storing data on various channels. In other words, from such a high level point of view, the concept of a channel is very similar to the concept of a space.

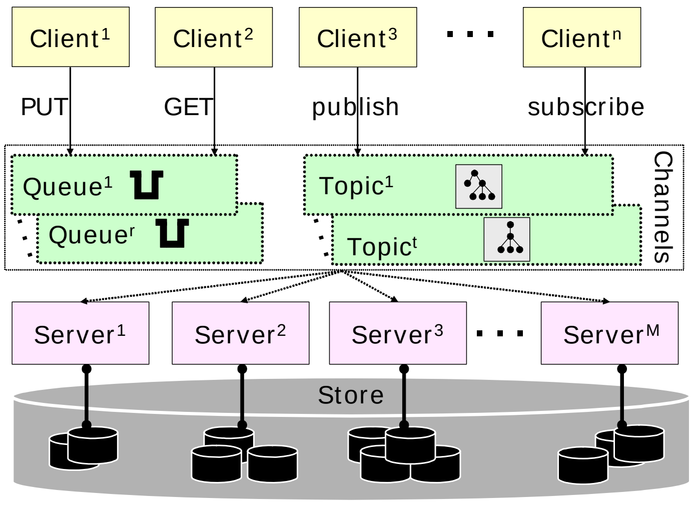

Figure 4 – Reliable Messaging Environment

Point-to-Point Channels - Channels can be very simple, allowing clients to store a message into the "space" (PUT) and to take an arbitrary message from the "space" (GET). Such simple kinds of channels are called point-to-point channels or queues; the name "queue" indicates the implementation technology underlying point-to-point channels. Note, that the purpose of a point-to-point channel is to make sure that a message is consumed by at most one target application. Consequently, all reads are destructive[4], i.e., point-to-point channels support "take" operations only, no "read" operations in terms of space technology. Point-to-point channels are very similar to spaces where all applications consume data from the space via "take" but not via "read."

Pubsub Channels - But a channel can also support simple semantic annotations of messages transported by it. For that purpose, semantic descriptions are modeled as so-called topic trees (e.g. [Happner et al., 2002]). A message is associated with a node of a topic tree (a so-called topic) indicating its semantics. Next, a message is written to a corresponding channel. Writing a message that is associated with a topic into a channel is referred to as "publish" (i.e. a message is "published on a topic"). Reading a message is a two-phase process: First, an application registers its interest in receiving certain messages by specifying topic-based filter expressions that possibly interesting messages must satisfy; registering interest is referred to as "subscribe." Second, when a message is published on a topic each subscriber with a matching subscription will receive a copy of the message; once each subscriber received its copy of the message, the message is destroyed from the channel. Corresponding channels are called publish-subscribe channels, or pubsub channels for short. Pubsub channels allow multiple applications to consume the same piece of data, which is similar to spaces allowing multiple applications to read the same piece of data. But each

4 For simplicity we ignore the ability to browse or peek messages in queues – which effectively is a read. But this is some sort of stretching the original usage pattern of queues.

application consumes a copy of the original piece of data written, and this consumption is destructive, i.e., it is a “take” but not via “read.” Thus, pubsub channels are very similar to spaces where all applications “read” data at most once.

Different Aspects of Autonomy - As the previous sections show, space environments and channel environments are very similar. Possible differences can be identified by studying certain autonomy aspects of the two. The following are considered important autonomy aspects of spaces (see [Bussler, 2005], [Fensel, 2004]):

- **Reference autonomy** denotes the fact that readers and writers in a space environment do not have to know each other; they simply exchange data via a space. The same is true for using a channel.
- **Space/location autonomy** refers to the property of spaces that readers and writers can be hosted by completely different environments, as long as they have access to the same space. The same is true for channels.
- **Time autonomy** allows readers and writers to access a space at their own pace, asynchronously, because the space maintains data persistence. The same is true for channels, with the caveat that the proper variants called “reliable message queuing” and “persistent pubsub” must be used.

Thus, even autonomy considerations can not really conceptually distinguish between spaces and channels, as is summarized in Table 1. Using spaces where data is taken (i.e. destructive read) from the space is not all to different from using queue based channels. Using spaces with non-destructive reads can not be distinguished from persistent pubsub according to the criteria used.

Approach	No. of readers	Time autonomy	Location autonomy	Reference autonomy
Space (take)	0..1	Y	Y	Y
Queue	0..1	Y	Y	Y
Space (read)	0..m	Y	Y	Y
Persistent pubsub	0..m	Y	Y	Y
Pubsub	0..m	N	Y	Y

Table 1 - Properties of Sample Communication Patterns

What is the Conceptual Difference? The last row of Table 1 reveals one conceptual difference between spaces and a certain kind of pubsub channel: non-persistent pubsub delivers copies of messages to all qualifying subscribers at the time the message is published. When a qualifying subscriber is not ready to consume the message at that time, it looses the message. Thus, non-persistent pubsub channels do not provide time autonomy. Persistent pubsub saves the message for later consumption: as soon as the subscriber is ready it will receive its copy of the message – time autonomy is back in place. Furthermore, a channel delivers a message once to each consumer: As soon as a message was consumed by a particular consumer the message will never be available again on the channel for this consumer. A particular consumer can never read the same message more than once from a channel.[5] Also, a channel pushes a message to a consumer, where within space environments a consumer pulls the data needed. Therefore, the conceptual difference between spaces and channels found so far is in repeatable reads. That is, one and the same application may read[6] (i.e. pull) the same tuple from a space multiple times, while each application may receive (i.e. push) a message from a channel it has access to only once.

Messaging/channel environments are reliable and production-ready. These environments have been invented and purposely built to be so. They support guaranteed delivery, transactions, fault handling, management, and so on. A comparison with existing space environments from this angle is worthwhile too. An important aspect is that of scale. Although messaging environments are proven to be highly scalable in practice (companies run environments distributed around the world, with hundreds of applications and many thousands of messages per second in production), Web-scale of messaging environments is something that the authors were not aware of. But the authors today's space environments were also unaware of current Web-scale demands. Thus, one of the key questions in this discussion is which technology to choose to extend with Web-scale capabilities. And this discussion must consider which quality of services each technology already currently supports.

The identified difference does not seem to justify introducing a new technology to the overall picture of application integration. Rich catalogues of patterns using channels have been developed over many decades of practical experience with channels. Each pattern describes how to add a particular behavior to a messaging environment (see [Hohpe & Woolf, 2004], for example). Thus, it is worth taking advantage of this large body of expertise by looking into patterns that allow multiple reads of the same message by the same application, and specifying the scenarios such behavior will be used for, as well as whether these would be scenarios targeted by messaging environments.

Vice versa, there are patterns that might be beneficial within space environments like “data type channels” exposing the message data type upfront, or the “command message”, “event message”, and “document message” patterns which express intent of a message upfront. Additionally, patterns like “return address” and “correlation identifier” could support a request-response kind of communication but in an asynchronous style.

[5] Of course messages with different identifiers can have the same message payload – but as usual, these are considered different messages.
[6] Not: take!

6. Conclusions

In his very stimulating paper, [Gray, 1995] pointed out that in principle, message queues are nothing else than simple databases for persistently storing and manipulating specific types of data, i.e., messages. However, he also pointed out the major weakness of his argument on a pragmatic level. He must presume that the environment is homogeneous. For a heterogeneous environment the costs of a full-blown database management system that would provide message queuing service is prohibitive. And this is where space-based communication enriched by semantics can add the missing piece to his lacking argument. Semantics-aware spaces are precisely a means to store semi-structured information accessed through a distributed and heterogeneous environment enabling to replace (or conceptually and functionally provide) a message queue with decent costs. Spaces offer an abstraction layer over message queues. Communication is abstracted by simply reading and writing data in a shared space. This allows for easier implementation of complex MEPs, especially if many parties are involved, if dynamic collaboration is needed, and if data needs to be read multiple times. A queue can already be considered a proven space-based pattern that supports FIFO interaction; whereas a space allows for extensions of this pattern towards specific selection from the queue, and retrospective changes of queued data.

However, combining just the concepts of space, semantics and Web services does not result into something fundamentally new. Much of the basic functional properties that space-based Web services reveal can be achieved via proper bindings exploiting established reliable messaging technology. It is not obvious at all, whether there are significant Semantic Web service applications that require the few missing functional properties resulting from being space-based. Furthermore, these properties may be realized based on a combination of bindings and MEP, i.e. based on established technology.

A different question is related to non-functional properties like reliability, availability, and scalability. Moreover meeting Web-scale requirements is critical and demands further investigations. These investigations have to answer the question whether reliable messaging technology or space technology is the right candidate for being enhanced towards Web-scale.

In the end, the difference may indeed be hidden since the access to data will be mediated through a semantic layer that is agnostic towards the underlying infrastructure. Systems illustrating how the two technologies could be successfully combined are already available and we expect to see more of them at large scale in the years to come. As an example [Petrie, 2006] describes how emailing could move from targeted message exchange towards publishing using concepts as addresses, whilst messaging is suddenly the same as publishing towards an anonymous audience.

Acknowledgements. This work was supported by the FP6 project TripCom (IST-4-027324-STP).

7. References

[Agha & Callsen, 1993]
G. Agha and C. Callsen. ActorSpaces: An Open Distributed Programming Paradigm. In *Proceeding of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1993.

[Berners-Lee et al., 2002]
T. Berners-Lee, J. Hendler and O. Lassila: The Semantic Web, *Scientific American*, May 2002.

[Bussler, 2005]
C. Bussler: *A minimal triple space computing architecture*, DERI Technical Report 2005-04-22, 2005.

[Cardoso & Sheth, 2006]
J. Cardoso and A. Sheth (ed.): Semantic *Web services, Processes and Applications*, Kluwer 2006.

[Fensel, 2004]
D. Fensel*: Triple-based computing*, DERI Technical Report 2004-04-31, 2004.

[Fielding, 2000]
R. T. Fielding: *Architectural styles and the design of network-based software architectures*, PhD Thesis, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[Fielding & Taylor, 2002]
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.

[Freeman et al, 1999]
E. Freeman, S. Hupfer, and K. Arnold: *JavaSpaces: Principles, Patterns, and Practice,* Addison-Wesley, 1999.

[Gelernter et al, 1985]
D. Gelernter, N. Carriero, S. Chandran and S. Chang: *Parallel programming in Linda*. In *Proceedings of the International Conference on Parallel Processing ICPP*, IEEE, 1985.

[Gelernter, 1989]
D. Gelernter: Multiple tuple spaces in Linda. In E. Odijk, M. Rem, and J.-C. Syre (eds.), *PARLE '89, Vol. II: Parallel Languages*, LNCS 366, pages 20-27, 1989

[Gray, 1995]
J. Gray: *Queues Are Databases*, Technical Report MSR-TR-95-56, 1995.

[Happner et al., 2002]
M. Hapner et al.: *Java Messaging Service API Tutorial and Reference*, Addison-Wesley 2002.

[Hendler et al., 2002]
J. Hendler, T. Berners-Lee, and E. Miller: *Integrating Applications on the Semantic Web*, J. Institute of Electrical Engineers of Japan, 122(10), 2002, p. 676-680.

[Hohpe & Woolf, 2004]
G. Hohpe and B. Woolf: *Enterprise Integration Patterns*, Addison-Wesley 2004.

[Khushraj et al., 2004]
D. Khushraj, O. Lassila, and T. Finin: *sTuples : Semantic Tuple Spaces*. In Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services – MobiQuitous'04, Boston, USA, August 2004.

[Kühn, 1994]
E. Kühn: *Fault-Tolerance for Communicating Multidatabase Transactions*. In Proceedings of the 27th Annual Hawaii International Conference on System Sciences (HICSS-27), Maui, Hawaii, USA, January 4-7, 1994.

[Kühn, 2001]
E. Kühn: Virtual Shared Memory for Distributed Architecture, Nova Science Publishers, 2001.

[Kühn et al. 2005a]
E. Kühn, J. Riemer, G. Joskowicz: XVSM (eXtensible Virtual Shared Memory) Architecture and Application, Technical Report TU-Vienna, E185/1, Space Based Computing Group, 2005.

[Kühn et al., 2005b]
E. Kühn, M. Beinhart, M. Murth: Improving Data Quality of Mobile Internet Applications with an Extensible Virtual Shared Memory Approach. In *Proceedings of the IADIS WWW/Internet 2005 Conference*, Lisbon, Portugal, October 19-22, 2005.

[Lehmann et al., 1999]
T.J. Lehmann, S.W. Mc Laughry, and P. Wyckoff: T *Spaces: The next wave*. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences (HICSS-32)*, Maui, Hawaii, USA, January 5-8, 1999.

[Leymann, 2005]
F. Leymann: The (Service) Bus: Services Penetrate Everyday Life. In *Proceedings of the 3rd Intl. Conf. on Service Oriented Computing ICSOC'2005*, Amsterdam, The Netherlands, December 13 – 16, 2005. Appeared as Lecture Notes in Computer Science (LNCS) 3826, Springer 2005.

[Menezes & Wood, 1999]
R. Menezes and A. Wood: Distributed Garbage Collection of Tuple Space in Open Linda Coordination Systems. In *Proceedings of the 14th International Symposium on Computer and Information Sciences,* Kusadasi, Turkey, October 1999.

[Petrie, 2006]

C. Petrie: Semantic Attention Management, *IEEE Internet Computing*, September 2006.

[Pinakis, 1992]
J. Pinakis. Providing Directed Communication in Linda. In *Proceedings of the 15th Australian Computer Science Conference*, 1992.

[Riemer et. al, 2006] J. Riemer, F. Martin-Recuerda, Y. Ding, M. Murth, B. Sapkota, R. Krummenacher, O. Shafiq, D. Fensel and eva Kühn: Triple Space Computing: Adding Semantics to Space-based Computing. In *Proceedings of the 1st Asiatic Semantic Web Conference (ASCW-2006),* LNCS, 2006.

[Tolksdorf, 1998]
R. Tolksdorf. Laura: A Service-Based Coordination Language, *Science of Computer Programming,* 31:359--381, 1998.

[Waldo, 1999]
J. Waldo: The Jini Architecture for Network-centric computing, *Communications of the ACM*, 42(7): 76--82, 1999.

[Weerawarana et al., 2005]
S. Weerawarana, F. Curbera, F. Leymann, T. Storey and D.F. Ferguson: *Web services Platform Architecture*, Prentice Hall, 2005.

[WSIF]
WSIF JMS Binding,
http://ws.apache.org/wsif/providers/wsdl_extensions/jms_extension.html #N10062

[Wyckoff et al., 1998]
P. Wyckoff, T.J. Lehmann, S.W. McLaughry, and D.A. Ford: *T Spaces*, IBM Systems Journal 37(3) 1998.

[zur Muehlen et al., 2004]
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.