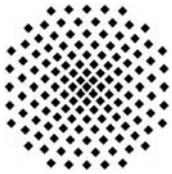

Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Space-Based Computing and Semantics: A Web Service Purist's Point-of-View

Frank Leymann

Report 2006/05
March 16, 2006

Institut für Architektur von Anwendungssystemen

Universitätsstr. 38
70569 Stuttgart
Germany

Space-Based Computing and Semantics: A Web Service Purist's Point-of-View

Frank Leymann
Institute of Architecture of Application Systems (IAAS)
Leymann@iaas.uni-stuttgart.de

Abstract: We review the concept of space-based computing and its combination with semantics into triple-space computing. Comparing this concept with existing technology we conclude that applying triple-spaces to Web service technology does not result into something fundamentally new to the architecture of Web services: we argue that this is an application of the concepts of binding and discovery which in contrast are fundamental aspects of the architecture of Web services.

1 Triple-Space Computing

There is a lot of interest in combining semantic Web technology (e.g. [2]) with Web service technology (e.g. [14]). The hope is to make discovery of Web services or even compositions of Web services much easier and more precise (e.g. [3], [8]). By adding semantics to Web services a requestor of a service may specify the service needed in terms of its problem domain, i.e. in business terms. This will be a major advancement compared to today, where discovery of Web services is mostly based on signature matching or just by QName of an implementation of a port type needed, or simply just by "knowing" the address of the port to use (i.e. its EPR - [14]).

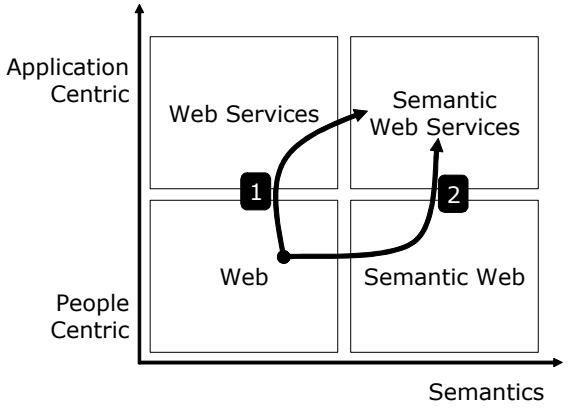

Figure 1 - Paths to Semantic Web Services [4]

Figure 1 depicts the phases in which the Web evolves [4]: The pure Web is centered on supporting human beings sharing information but 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, 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 not by human beings directly; as a consequence, semantic descriptions have not been considered at all when this technology appeared. Semantic Web services are Web services enriched by semantic descriptions to make them discoverable more easily and much more precisely.

The technology supporting semantic Web services can be evolved along two paths (Figure 1): First, Web service technology can be extended with semantics, i.e. it can be combined with corresponding technology from the field of semantics; 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 as envisioned in [4] is about the second path: it strives towards applying the concept of “spaces” (see [15], e.g.) to the semantic Web [11] – which is why this combination is also referred to as triple-space computing in [1].

1.1 The Concept of Space

Space computing has its origin in parallel programming [6]. 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 (“write”). Any number of components can read a piece of data without removing it from the space (“read”); a component may read the same piece of data even multiple times. There is also a variant of reading data that removes data from the space once it is read (destructive read – referred too as “take”).

A space stores data persistently such that it is not lost when the environment realizes an outage. The data model supported by a space is that of a tuple, which is why a corresponding space is also referred to as tuple space (“TSpace” [15]). In a TSpace, 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 one will be returned from the space. To support more sophisticated applications, advanced features for manipulating tuples in a space are defined (e.g. [12]). Finally, matching of tuples to be retrieved from a space can be supported by associating semantics with each tuple (so-called “sTuples” [11]).

1.2 Web Architectural Style and Spaces

As observed in [4], 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 [5]. Identifying resources in the Web by means of URIs is seen as one of the underpinning of the architectural style behind the Web (the so-called REST style [5]) and it is the major ingredient for scalability of this architectural style 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. But access to tuples via URIs is cumbersome, access via semantic queries is much more convenient.

In the Web, semantics is associated with resource based on RDF by specifying a *triple*: namely by assigning an “object” via a “predicate” to a “subject” (the resource). Thus, what is to be written or read from this kind of a space is a

triple: a semantically enriched tuple where the tuple is the RDF resource[1]. Tuples can be discovered by semantic queries returning the URIs of the tuples qualified, and based on these URIs the tuples can be accessed.

Another underpinning of the architectural style of the Web is that of a very simple set of methods for accessing resources [5]: basically, four methods are provided that allow to create, read, update, and delete a resource (so-called CRUD methods). These four methods are the key methods supported by HTTP. In [1] 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.

1.3 Space Environment Architecture

Figure 2 depicts the triple space environment from a very high level architecture point of view. 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 [1]). The overall environment consists of multiple spaces (like the Web is made up of multiple domains). The property of a space of being persistent 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 storing a fragment of a subset of all spaces (like Web servers store the resources manipulated via HTTP requests).

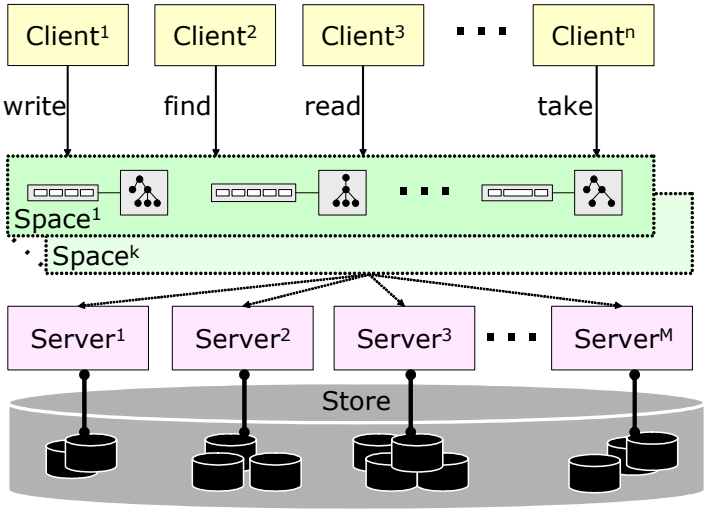

Figure 2 – Triple Space Environment

2 Reliable Messaging Technology

One application area envisioned to become of fundamental importance for semantic Web technology is application integration [8]. Figure 3 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. [10]). The predominant technological underpinning of integration environments is reliable messaging.

[1] This is a bit confusing because mathematically, each triple is a tuple.

2.1 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.

I.e. from that very high level point of view the concept of a channel is very similar to the concept of a space.

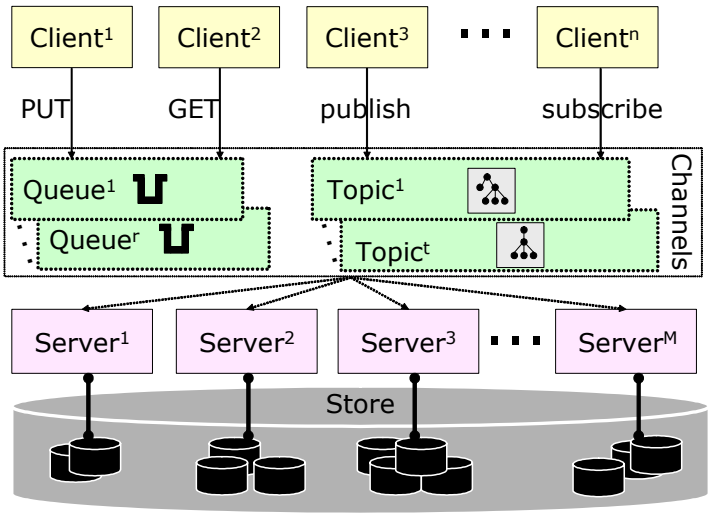

Figure 3 – Reliable Messaging Environment

2.2 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[2], 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".

2.3 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. [7]). 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

[2] 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.

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 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 ones.

2.4 Different Aspects of Autonomy

As worked out so far, space environments and channel environments are very similar. Trying to get closer to possible differences between spaces and channels autonomy aspects are discussed. The following autonomy aspects are considered important aspects of spaces (see [1], [4]):

- “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 persists data. 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 distinguish spaces and channels conceptually – which is summarized in Table 1: Using spaces where data is taken (i.e. destructive read) from the space is not 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

2.5 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 got 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[3]. Also, a channel pushes a message to a consumer, where within space environments a consumer pulls the data needed.

Thus, the *conceptual* difference between spaces and channels found so far is in repeatable reads: One and the same application may read[4] (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.

2.6 But this is All About Patterns

The difference identified does not seem to justify introducing 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 particular behavior to a messaging environment (see [10], for example). Thus, it is worth investigating a pattern that allows multiple reads of the same message by the same application – although this investigation should be accompanied by developing scenarios for what purposes such behavior will be used for and whether these would be scenarios messaging environments are targeting for.

Vice versa, there are patterns that might be beneficial within space environments like “data type channels” enabling to know the type of data of a message upfront. Or the “command message”, “event message”, and “document message” patterns allowing to express the kind of intend of a message. Or pattern like “return address”, and “correlation identifier” that support a request-response kind of communication but in an asynchronous style.

2.7 What About QoS?

Messaging/channel environments are reliable, production-ready: these environments have been invented, have been 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 author is not aware of. But neither is the author aware of Web-scale of today’s space environments.

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

Thus, one of the key questions in this discussion is which technology to choose to extend with Web-scale capabilities. And this discussion must take into consideration which quality of services each technology does support today already.

3 Message Exchange in Web Service Technology

Web service technology has been invented to solve a couple of hard application integration problems [14]. Application integration implies that messages are to be exchanged between applications. Message exchange requires transporting messages from the sender to the receiver.

3.1 Bindings and Transports

Within enterprises transport mechanism like JMS or RMI/IIOP, for example, are of much higher importance w.r.t. application integration than HTTP. Because of that, 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 (WSDL) *binding*. In a nutshell, a binding specifies how a message is sent from a source to a target (the service), i.e. 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.

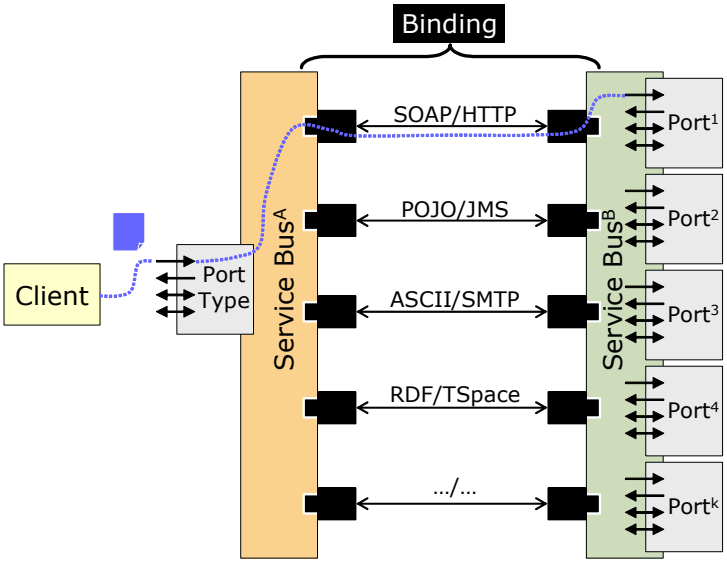

Figure 4 - Bindings Enabling Communication via Multiple Transports

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 – i.e. the service bus performs service *discovery* on behalf of the requesting client.

Typically, a binding corresponds to a handler, i.e. a plug-in of the service bus understanding how to use the particular transport protocol etc. – see Figure 4. Note, that it is absolutely valid (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 (like policies, service level agreements, workload considerations etc. [13]).

3.3 Semantics and Discovery

Typically, service discovery is not only based on the interface the service must support but also on additional criteria like QoS or SLAs (e.g. support of transactions, privacy behavior, availability, response time, etc. – specified via appropriate policies [14]). The service bus uses all this information to determine a list of services that satisfy all the criteria. If more than one service qualifies, additional criteria are applied to actually select the service to be used to perform the client's request. Effectively, the service bus *virtualizes* all services qualifying under the *declarative request* of the client (note the interesting similarity to declarative database manipulations here). Today, the criteria used for discovery are IT centric. Annotating Web services with semantics will allow to add semantics to the selection criteria making determining target services more precise in terms functionality performed etc.

But extending discovery capabilities by semantics and extending the set of available bindings by space-concepts are orthogonal threads of investigations.

3.4 Potentially Misleading: "Web" in "Web Services"

As the concept of a binding reveals, taking the prefix "Web" in "Web service" too serious may be misleading, stimulating 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 from a technical point of view and more correct.

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 (e.g. transmitting Java objects over JMS is an often used binding within Web service applications – see [16] for more detail about this and other bindings).

3.5 Bindings and Spaces

The before-mentioned JMS binding has interesting properties when compared with an HTTP binding: (1) 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; (2) 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; (3) the JMS binding could support publishing a message on a topic, i.e. multiple recipients could get (copies of) the message; (4) the JMS binding support 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 [4]. As discussed above (section 2.5), 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: (1) define the Web service scenarios that require repetitive receipt of the same message by a given Web service; (2) define a “space binding” for Web services, i.e. a binding that uses space technology for message exchange in Web service communication like the JMS binding uses queue technology or pubsub technology for the same purpose.

3.6 Message Exchange Patterns

An operation of a Web service always follows a message exchange pattern [14]. 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, second, expects multiple messages of another type back.

Note, that an 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. But it is obvious that proper bindings ease the implementation of an 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 Summary

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 that of non-functional properties like reliability, availability, and scalability. Especially the requirement of Web-scale 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.

5 Acknowledgements

The first one who got me in touch with space-based computing many years ago was Eva Kühn – in a completely different context, I am sure she will recall. Dieter Fensel, years later, discussed with me the parallelism between the evolution the Web took and what happened in Web service technology at that time, and that the concept of space could be a common abstraction important for both venues. Chris Bussler and I had some discussions on various alternatives to implement the concept of spaces with Web scale. I am grateful to all three of them for these discussions that influenced my thinking about the subject matter.

6 Literature[5]

[1] Ch. Bussler: A minimal triple space computing architecture, DERI Technical Report 2005-04-22, 2005.
[2] T. Berners-Lee, J. Hendler and O. Lassila: The Semantic Web, Scientific American, May 2001.
[3] J. Cardoso and A. Sheth (ed.): Semantic Web Services, Processes and Applications, Kluwer 2006.
[4] D. Fensel: Triple-based computing, DERI Technical Report 2004-04-31, 2004.
[5] R. T. Fielding: Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[6] D. Gelernter, N. Carriero, S. Chandran and S. Chang: Parallel programming in Linda, Proc. Int'l Conf. on Parallel Processing ICPP, IEEE, 1985.
[7] M.Hapner, R.Burridge, R.Sharma, et. al.: Java Messaging Service API Tutorial and Reference, Addison-Wesley 2002.
[8] J. Hendler, T. Berners-Lee, E. Miller: Integrating Applications on the Semantic Web, J. Institute of Electrical Engineers of Japan, 122(10), 2002, p. 676-680.
[9] M. Hepp, F. Leymann, J. Domingue, A. Wahler and D. Fensel: Semantic Business Process Management: Using Semantic Web Services for Business Process Management, Proc. IEEE ICEBE 2005 (Beijing, China, October 18-20, 2005).
[10] G. Hohpe and B. Woolf: Enterprise Integration Patterns, Addison-Wesley 2004.
[11] D. Khushraj, O. Lassila and T. Finin: sTuples : Semantic Tuple Spaces, Proc. 1st Annual Int'l Conf. on Mobile and Ubiquitous Systems: Networking and Services – MobiQuitous'04 (Boston, USA, August 2004).
[12] T.J. Lehmann, S.W. McLaughry and P. Wyckoff: T Spaces: The next wave, Proc. 32nd Annual Hawaii Int'l Conf. on System Sciences HICSS-32 (Maui, Hawaii, USA, January 5-8, 1999).
[13] F. Leymann: The (Service) Bus: Services Penetrate Everyday Life, Proc. 3rd Intl. Conf. on Service Oriented Computing ICSOC'2005, (Amsterdam, The Netherlands, December 13 – 16, 2005), LNCS 3826 Springer 2005.
[14] S. Weerawarana, F. Curbera, F. Leymann, T. Storey and D.F. Ferguson: Web Services Platform Architecture (Prentice Hall, 2005).
[15] P. Wyckoff, T.J. Lehmann, S.W. McLaughry, and D.A. Ford: T Spaces, IBM Systems Journal 37(3) 1998.
[16] WSIF JMS Binding, http://ws.apache.org/wsif/providers/wsdl_extensions/jms_extension.html#N10062

[5] Links followed on 3/10/06