Distrib Parallel Databases (2006) 20:91–115
DOI 10.1007/s10619-006-9447-1

Integration of transient Web services into a virtual peer to peer Web service registry

Schahram Dustdar · Martin Treiber

Published online: 28 July 2006

Abstract Transient Web service provisioning implies a variety of different requirements that are hard to meet in traditional Web service environments. Currently, Web service brokerage focuses on centralized or replicated architectures. We argue that such systems are not efficient when it comes to dynamic, respectively ad hoc, Web service provisioning. We propose a distributed peer to peer Web service registry solution based on lightweight Web service profiles. We further introduce the notion of views that allow the specification of arbitrary contexts of Web services and provide a working example to illustrate our approach. Finally, we present a prototype that uses tuple spaces as global storage and communication means.

Keywords Web services · Pervasive computing · Tuple spaces · Web service registries

1. Introduction

Web services [27] are becoming a very important means to architect and integrate distributed applications. The W3C defines a Web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [16–18]). Other systems interact with the Web service in a manner prescribed by its description using SOAP [14, 15] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Generally speaking, a Web service provides a public interface that is described in an interface language, for example WSDL [16–18]. In order to find Web services, Web service

Recommended by: Ahmed Elmagarmid

S. Dustdar (✉) · M. Treiber
Distributed Systems Group, Vita Lab, Institute of Information Systems,
Vienna University of Technology
e-mail: dustdar@infosys.tuwien.ac.at

M. Treiber
e-mail: m.treiber@infosys.tuwien.ac.at

registries are needed. Web service registries are databases where Web service descriptions are stored. Current Web service architectures [10] provide registries that usually are based on centralized systems. Examples include UDDI [1] and ebXML [2–5] which provide centralized registries for Web service brokerage.

Because of the growing availability of public wireless networks it is possible to use Web services from locations outside of office buildings and to virtually extend the working environment to public places. The use of Web services in such dynamic environments is well established. However, little attention is paid to *Web service provisioning* in such a dynamic environment where network connections may exist only for a few minutes or hours. We call this kind of Web service provisioning *transient Web service provisioning* and the corresponding Web service provider *transient Web service provider*.

Consider for example a conference meeting and a feedback Web service. Conference participants may be interested in immediate feedback about presentations, workshops, tutorials, etc. During the conference the feedback Web service is available to other conference participants. It can be the case that a Web service provider leaves the conference and returns afterwards. During the absence of the conference, the Web service provider is not interested in providing the feedback Web service.

With this type of Web service provisioning arises a set of requirements for transient Web service providers, respectively, for transient Web service provisioning:

- *Accuracy of Web service registries.* Under some circumstances, like, for example, instable network connections, notifications concerning the availability of Web services may not be received by the Web service registry. This leads to Web service registry entries without actual Web service provider. We believe that such a state of registry entry—with inaccurate Web service registry entries—is not desirable.
- *Tight coupling between Web service provider and Web service.* We believe that there is a tight coupling between Web service provider and Web service offering, because the behavior of transient Web service providers is reflected by the availability of their Web services.
- *Distributed Web service registry.* In environments that are defined by transient Web service providers it is important that Web service registries offer a scaleable and flexible approach for accessing Web service registry data.
- *Loose coupling of messages.* Due to disconnections from the network, transient Web service providers need a messaging system that decouples the sender from the receiver.
- *Context information.* In order to be aware of transient Web service provisioning and Web service communities Web service registries need context information.

Although current registries acknowledge some of the aforementioned requirements and provide, for example, registry replication and structures for context information, they do not support transient Web service provisioning. With our work, we aim to move Web service registry technology towards transient Web service providers and to provide context information for Web service registry entries. We developed VISR (**V**iew based **I**ntegration of Web **S**ervice **R**egistries)—a peer to peer architecture for distributed Web service registry, that meets these requirements.

In particular, our research prototype addresses the following issues:

- *Distributed Web service registry.* To circumvent inaccurate registry information, we propose a distributed Web service registry solution that uses tuple spaces [19] as primary communication and publication means. The tuple space concept allows for higher flexibility concerning transient Web service provider. Tuple spaces use simple primitives (in, out, etc.) for the creation, removal and reading of information. We choose tuple spaces for the creation of a virtual distributed registry due to its small administrative overhead.

- *Arbitrary declarative context for Web service description.* We propose an abstraction (VISR view profiles) that acts as container for context-information. We consider Web service communities as primary target for context information. VISR view profiles provide the means for the creation of virtual communities and serve as means to structure global Web service registry content.
- *Common Web service description.* We introduce VISR service profiles that serve as lightweight description language for different Web services registries. We acknowledge existing Web service registry data models (UDDI, ebXML) and provide transformations of existing registry content to our data model. Our intension is to hide the heterogeneity of different Web service registry data models and to provide a common lightweight Web service description.
- *Integration of transient Web service providers.* Transient Web service providers are volatile members of a Web service network. Our work considers the requirements of transient Web service providers under the aspect of accuracy of Web service registries and administrative overhead. VISR's tuple space based registry implementation supports dynamic joining and leaving of Web service networks without centralized registry mechanisms.
- *Decoupling of messages.* Related to transient Web service provision is the decoupling of messaging in a dynamic environment. We use tuple space operations for the decoupling of messages between Web service requestors and Web service providers.
- *Structural matching of Web service description.* We present a simple algorithm for matching of structural similarities (SSMA) of Web service descriptions specified as VISR service profiles.

The rest of the paper is organized as follows: Section 2 briefly summarizes the existing Web service registry approach and introduces the concept of the VISR peer to peer registry. Section 3 presents the architecture of VISR. The implementation details are discussed in Section 4. Section 5 gives a description of the discovery algorithm that is used in VISR. Section 6 illustrates a working example. Section 7 lists related work and Section 8 concludes the paper.

2. Distributed virtual peer to peer registry

This section discusses VISR's distributed virtual Web service registry. We move the traditional Web service paradigm towards a distributed solution. In the traditional Web service paradigm, a Web service broker manages the Web service registry entries (Fig. 1).

The concept of a dedicated Web service registry entity decouples Web service providers from the storing the actual Web service description. We believe that this concept proves to be inflexible in a dynamic environment with transient Web service provision.

Generally speaking, transient Web service providers offer their Web services for limited time. This may be because of the membership to an ad-hoc community, which may be specified for instance for a meeting, or because of temporal limited network connections using Wireless LAN in public places. Therefore, transient peers depend on a Web service registry concept that supports decoupled (Web service registry) operations. Centralized Web service registries are not practical when they contain entries that point to unavailable services. Therefore, we intend to transform the Web service brokerage model to a distributed model with implicit Web service brokerage to provide accurate Web service registry entries. We

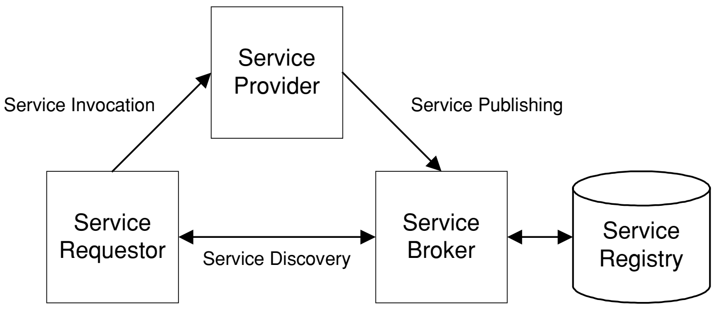


Fig. 1 Conceptual overview of Web services in the service oriented architecture

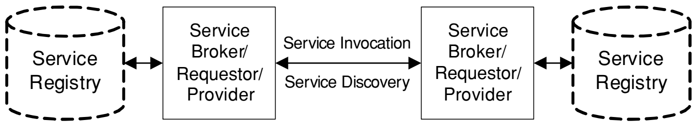

Service Registry
Service Broker/ Requestor/ Provider
Service Invocation
Service Discovery
Service Broker/ Requestor/ Provider
Service Registry

Fig. 2 Conceptual overview of VISR's Web service registry paradigm. The dotted lines mark parts of the distributed virtual registry. Every Web service provider is at the same time Web service broker, requestor and provider

propose a peer to peer based Web service registry solution where every transient peer is implicitly a part of the virtual Web service registry. In this way, we move the traditional Web service registry paradigm towards peer to peer systems. Figure 2 shows the concept of VISR's Web service registry.

VISR's distributed registry model is based on the Blackboard architectural pattern [33]. It allows VISR peers to operate on a common data structure and provides a common vocabulary. This paradigm provides useful features for distributed and dynamic environments:

- It supports Web service publishing and discovery in a distributed space. This interaction style is useful in dynamic environments where it is very important to reduce administrative overhead to a minimum.
- Publishing and Web service invocation can be executed without prior knowledge of other management entities. It allows flexible modeling of interactions among services in highly-dynamic environments.

Example. Ad-hoc meetings are an example for the use of distributed virtual Web service registries. Ad-hoc meetings define Web service communities that exist over a specific period and provide a certain set of services. During the ad-hoc meeting it may be necessary to share data (presentations, pictures, documents, etc.). To accomplish this, a file sharing Web service is offered where peers are can share their files. The Web service is offered as long the peer is a member of the community, i.e. participates in the meeting. After the peer leaves, the Web service is not available any more, because the peer does not want to share documents outside the meeting.

Centralized Web service registries are not useful in such scenarios, because in order to provide a service each provider must register the service prior to the meeting. Upon leaving, the Web service provider must notify the Web service registry that the service is not available anymore. If the Web service provider rejoins the community, the Web service registry must

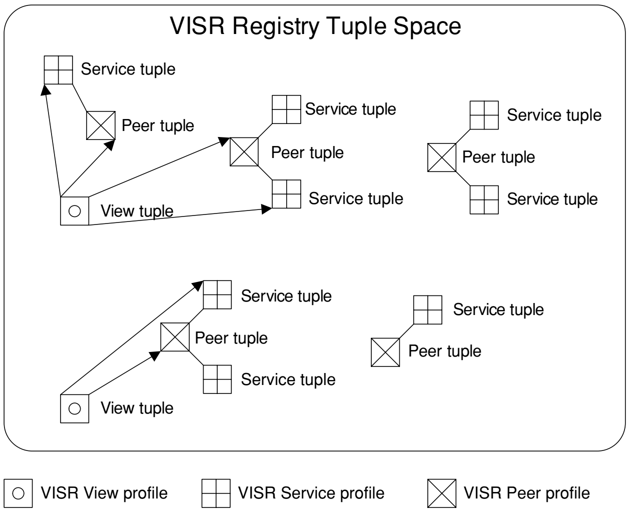


Fig. 3 Conceptual overview of VISR Web services registry model. Arrows mark membership with a VISR view respectively with a community. Note, that not every service of a VISR peer is a member of view

be updated again in order to provide accurate Web service descriptions. The centralized approach illustrates the typical bottleneck of single Web service registries.

In order to provide a flexible data model for distributed Web service registries VISR models three different types of registry information. VISR distinguishes between

- Information about providers (VISR peer profiles) that includes data about the Web service provider such as memory, CPU, maximum supported Web services, name, contact info, etc. This kind of information is used to identify Web service providers.
- Context information (VISR view profiles) that provides context information about Web services and serves as means to organize the global Web service registry content within Web service communities.
- Web service descriptions (VISR service profiles) that provide a lightweight Web service description.

Figure 3 shows a conceptual overview of the distributed Web service registry structure. VISR uses a tuple space to provide a global common data structure. The global tuple space holds tuples that contain VISR view, VISR service and VISR peer profiles.

VISR view profiles. VISR view profiles serve two purposes, they (i) provide the means to structure the content of the global Web service registry tuple space and (ii) provide contextual information about Web services.

A global Web service registry tuple space offers no means for structuring of content. VISR views provide the means to organize Web services into Web service communities. Web service communities can be regarded as cluster of Web service (provider) that share common interests, like for example the membership in a project team, etc. The creation of views consists of the publishing, i.e., writing, of VISR view profiles into the shared tuple space. VISR peers that are interested in the view must implement the operations specified by the view. These descriptions are part of the operational portion of views. VISR views do not provide typing or binding information. This is left to the VISR peer that must implement

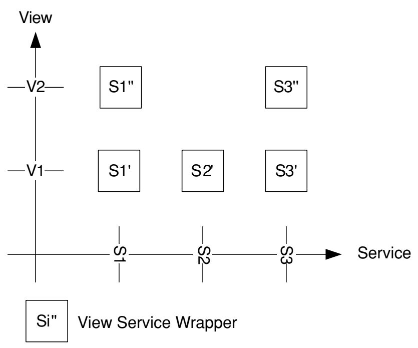


Fig. 4 VISR views. Concrete service instances are depicted on the *x*-axis. Views are depicted on the *y*-axis. Every view provides a different context for Web services and defines different interfaces for Web service. In the example, services S1, S2 and S3 provide different interfaces. Web service provider use local transformations—S1′, S2′ and S3′ for View V1 respectively S1″ and S3″ for View V2 to provider view compliant Web services

the actual transformation of the data when calling a Web service. VISR supports this process with the help of Wizards that assist the user in the manual creation of mappings (see Section 4).

In order to provide additional (context) data VISR views provide context information. Context information includes for example membership in Web service communities, information about requirements concerning Web service provider (Disk space, CPU, etc.), and so on. This type of meta data provides interested peers with additional information. This information is usually out of the scope of single Web service descriptions, since the context of Web services is subject to change. For example, when joining a community, a Web service context is implicitly created by local data transformations. VISR views encapsulate these local transformations with the help XSLT [12] expressions. The example below shows a transformation of a "legacy" notification Web service that uses one input parameter and one output parameter into a community notification service that provides no output at all.

```
<Mapping>
  <!-- the ''legacy'' method is called Notify ->
  <Element type=''method''>
    <Name>Notify</Name>
    <!-- the actual transformation is an XSL expression ->
    <Expression>
    ..
    <xsl:for-each select="Method/Paramters/Parameter/">
       <xsl:if test="@type='Input'">
         <Name><xsl:value-of select="Name"/></Name>
       </xsl:if>
    </xsl:for-each>
    ..
    </Expression>
  </Element>
</Mapping>
```

Figure 4 shows different contexts of Web services within different views. This approach enables Web services to be part of several views at the same time without the need of any change in the Web service description.

VISR service profiles. VISR service profiles provide a common generic lightweight and extendable Web service description. VISR service profiles embrace standards such as UDDI and WSDL. VISR service profiles extend existing Web service descriptions with abstract operational descriptions that enables operational driven Web service discovery. Furthermore, VISR service profiles provide usage scenarios that consist of sequences of method invocations. Usage scenarios illustrate how the Web service is invoked and what is returned by a Web service. Note that VISR Service profiles do not contain actual bindings or typing information. The actual binding is left to the VISR peer that provides the Web service.

VISR peer profiles. VISR peer profiles model information about VISR peers. They consist of data about the nature of the peer (transient or static). Furthermore, information about capabilities are modeled, for instance, the number of Web services supported and information that includes technical data such as port numbers and information about the provider including for example name, organization, etc.

3. VISR architecture

This section discusses the layered architecture of transient VISR. VISR's architecture is divided into three layers (registry, view and service) as shown in Fig. 5. The registry layer provides the operations for the administration of VISR registry entries. Its main task is the persistency-management of VISR profiles. On top of the registry layer operates the view layer which provides operations needed for the view based transformation of VISR service profiles. Using the view layer, the service layer provides interfaces for basic functionality of the VISR peer that include for example inter peer communication. The following subsections discuss these three layers in detail.

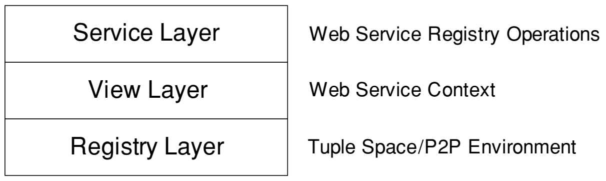
3.1. Registry layer

The registry layer consists of a tuple space based repository for VISR profiles. The registry layer provides the abstractions that let the registry appear as single logical block. It provides the basic functionality for the publishing, unpublishing and discovery of VISR service profiles, VISR view profiles and VISR peer profiles. The publishing/unpublishing of VISR service profiles consists of writing/deleting corresponding tuples into/from the shared tuple space. The discovery of Web services involves the matching of corresponding tuples of the tuple space. VISR supports two different methods for Web service discovery. VISR uses either XPath expressions, or a simple structure matching algorithm (discussed in Section 6) to select VISR profiles.

3.2. View layer

The view layer provides the means for the management of VISR views. The view layer consists of a local repository that stores VISR view descriptions and corresponding matching descriptions of VISR view filters to VISR service profiles. The view layer is responsible for

Fig. 5 VISR architecture



the actual invocation of Web services and transformation of requests according to VISR view descriptions. VISR's view layer parses the incoming requests and selects the corresponding Web service for execution. The view layer also transforms the results of the Web service into the expected format.

3.3. Service layer

The service layer provides the core functionality for the basic operations of the Web service registry, i.e., the publishing and unpublishing of Web services and provides meta data services. The latter include VISR view services that provide for example peer lists and VISR view descriptions. The service layer consists of an access interface and also exposes itself as a Web service with a corresponding VISR service profile. It receives and parses the incoming requests and forwards the requests to the view layer. After the request is parsed the corresponding operation is being executed. Therefore, the service layer contains a repository of services that are available at the VISR peer.

4. VISR prototype implementation

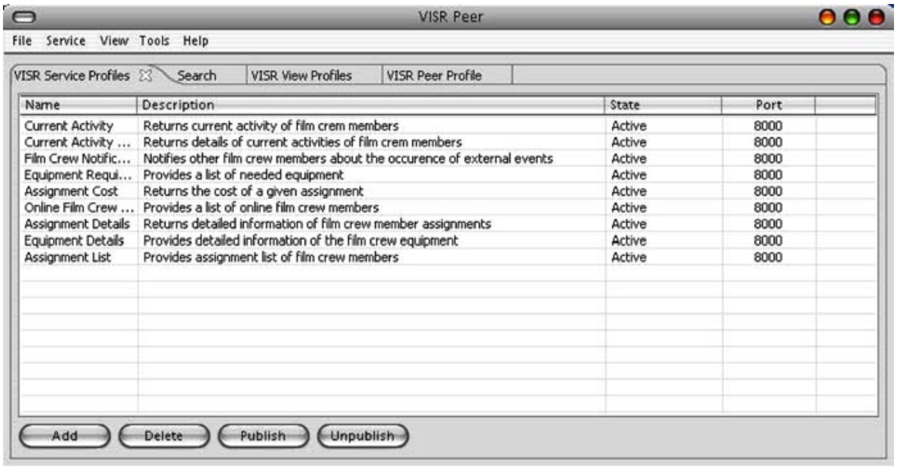
The VISR prototype was developed using IBM's tuple space implementation TSpaces [11] and Java 1.5 [24]. VISR provides Java objects that encapsulate VISR service profiles, VISR view profiles and VISR peer profiles (see Appendix A for the APIs). Tuples in the shared tuple space either represent VISR peer profiles, VISR view profiles, or VISR service profiles. These three types are instances of the abstract base class `VISRProfile`. Every instance of the class `VISRProfile` provides a standard method to represent the information stored in the profile as string. VISR extends TSpaces subclass able tuple definition and provides a generic container for VISR view, VISR service, and VISR peer profiles called `VISRProfile`. This abstract class extends the Boolean tuple matching function of TSpaces by implementing the interface `VISRMatchable`. The matching function returns a value either 0 (indicating no match) or 1 (indicating a potential match). This class hierarchy exposes similarities to the class hierarchy discussed in [6] that provides different matching functions for tuples.

Our prototype provides two basic communication interfaces, one interface for the view administration and one interface for the publishing/discovery of Web services. These two interfaces are described by VISR service profiles that are implemented using either SOAP messages for communication or tuples. VISR peers provide a default service port. This port intercepts all communication and serves as service port for the VISR service invocation.

Figure 6 shows the User Interface of a VISR peer client. The main window provides lists of available VISR Services at the client and shows basic information about services.

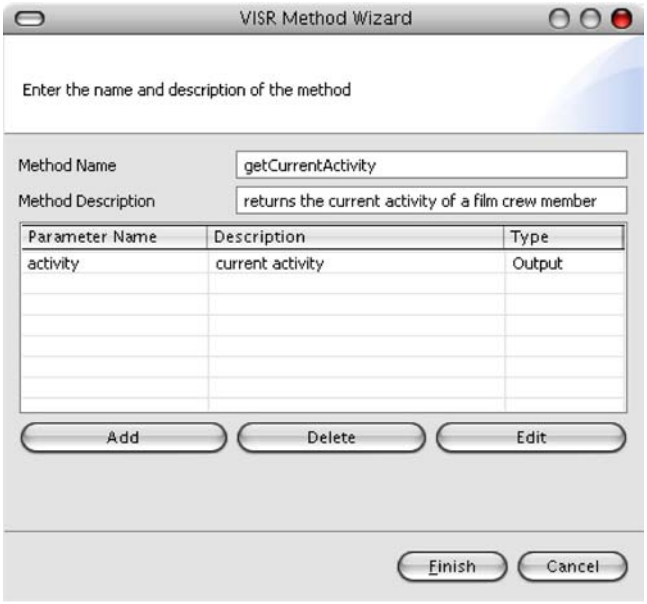
The creation of VISR Service profiles is guided by service creation Wizards, as depicted in Fig. 7. The user creates a VISR service profile by defining the name and the operations (methods, and corresponding parameters) of the Web service.

The creation of interface mappings (parameter, method names) for the use of Web services within different views is usually the last step in the process of creating a VISR compliant Web service. The user is also guided by Wizards that support the declaration of mappings for the messages. Figure 8 shows the interface for the creation of mappings. After the completion of method/parameter mappings the corresponding Web services can be used in different contexts (views) by other peers of the VISR network.



Name	Description	State	Port
Current Activity	Returns current activity of film crem members	Active	8000
Current Activity ...	Returns details of current activities of film crem members	Active	8000
Film Crew Notific...	Notifies other film crew members about the occurence of external events	Active	8000
Equipment Requi...	Provides a list of needed equipment	Active	8000
Assignment Cost	Returns the cost of a given assignment	Active	8000
Online Film Crew ...	Provides a list of online film crew members	Active	8000
Assignment Details	Returns detailed information of film crew member assignments	Active	8000
Equipment Details	Provides detailed information of the film crew equipment	Active	8000
Assignment List	Provides assignment list of film crew members	Active	8000

Fig. 6 VISR peer main window



Enter the name and description of the method

Method Name: getCurrentActivity

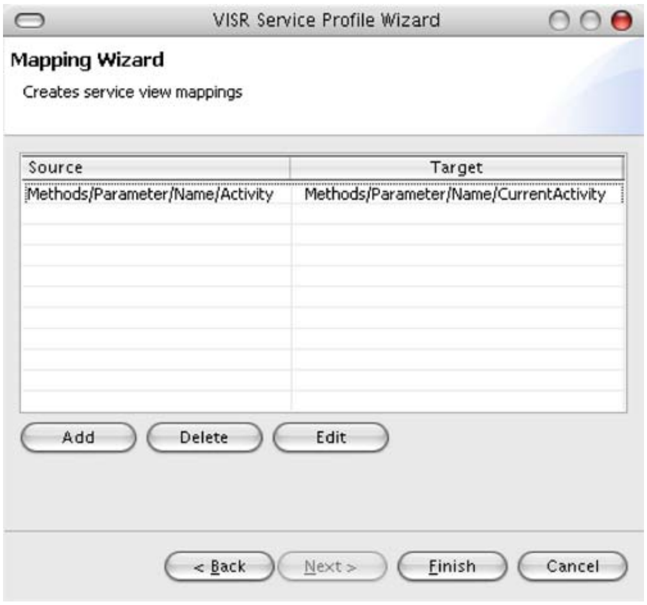
Method Description: returns the current activity of a film crew member

Parameter Name	Description	Type
activity	current activity	Output

Fig. 7 VISR method creation Wizard

4.1. VISR peer initiation protocol

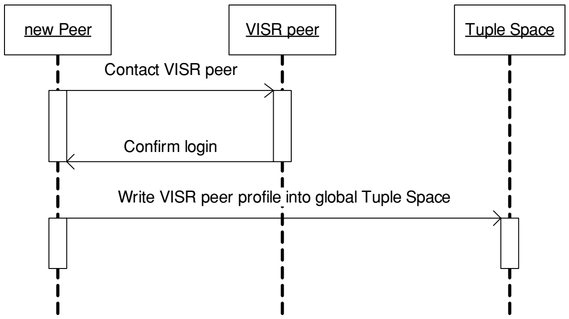
The initiation of a new VISR peer consists of writing a VISR peer profile into the global tuple space. In order to obtain a handle to the global tuple space, a new peer must contact a VISR peer that provides the new peer with a global tuple space handle. Afterwards, the new peer writes its service profile into the global tuple space and makes its Web services available. Figure 9 shows the initialization protocol of VISR peers.



Source	Target
Methods/Parameter/Name/Activity	Methods/Parameter/Name/CurrentActivity

Fig. 8 VISR Mapping Wizard. The example shows a basic mapping between a method parameter *Activity* (defined by a view) and the corresponding parameter with the name *CurrentActivity* of a VISR service profile

Fig. 9 VISR peer initialization protocol



If the peer wants to leave the network, it must remove its peer tuple from the tuple space. Every peer tuple contains a timestamp that must be renewed by the VISR peer periodically. The timestamp value depends on the actual application and can be defined by the VISR peer itself. If the timestamp is renewed, other VISR peers can assume that the peer is going to be available for another pre defined period of time. If the timestamp is not renewed within the pre specified time period, the tuple is removed from the tuple space. In addition, all service tuples of the peer are also removed from the virtual registry.

4.2. VISR peer alive signal

In order to provide information about the availability of VISR peers it is necessary for every VISR peer to renew its peer profile periodically. The validity period is defined by the VISR peer itself and depends on the context of the peer. If a VISR peer does not manage to renew its profile in the global tuple space in time, it is automatically removed from the tuple space.

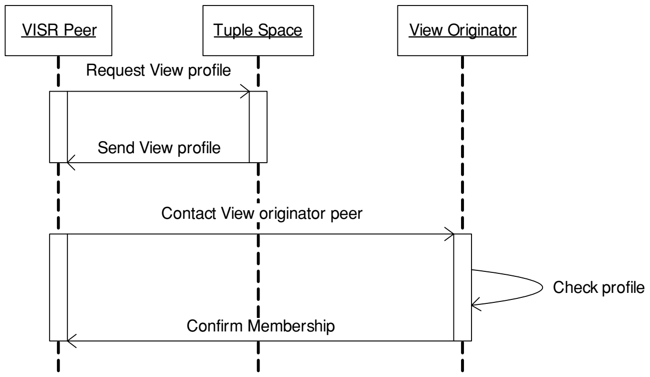


Fig. 10 VISR view initialization protocol

This may be for example the case if a VISR peer leaves the network abruptly, without removing the VISR peer profile.

4.3. VISR view initialization protocol

The creation of dynamic communities follows the pattern as depicted in Fig. 10. First, a VISR peer creates a view profile and writes it into the tuple space. In order to join a view, a VISR peer requests the corresponding VISR view profile. After the peer has obtained the VISR view profile, usually some service mappings have to be built in order to provide the services specified by the view. After this task is completed the VISR peer contacts the View originator and sends its profile. The view originator checks the profile and finally confirms the membership by updating the peer-list of view profile.

In case of being a member of a VISR view, the peer does not need to provide an alive-signal periodically. After registering for a view, VISR peers are automatically network members as long as the view exists.

4.4. VISR communication interface

VISR peers provide two standard communication interfaces. A SOAP based interface and a tuple space based interface. Both interfaces offer the same basic functionality, but a different degree of coupling. While SOAP messages need a message receiver that is available at the time a message is being sent, Tuple space based messages are completely decoupled from time and space. Furthermore, VISR's communication interface supports the view based Web service invocation, which uses context information for the invocation of Web services.

Decoupled operation. In this mode, the sender places a message tuple in the tuple space. VISR differentiates between to different kinds of messages, i.e., messages that are sent in the context of views and global messages without reference to views. The latter contain a reference to the corresponding view and are only retrieved by members of the view. The tuple also contains either a reference to the receiver of the message or a generic identifier for broadcast messages. As soon as the recipient is online again, the VISR peer is notified that a new message tuple awaits him and the VISR peer reads the message tuple from the tuple space. A message tuple has a validity period and is removed by the sender as soon as the

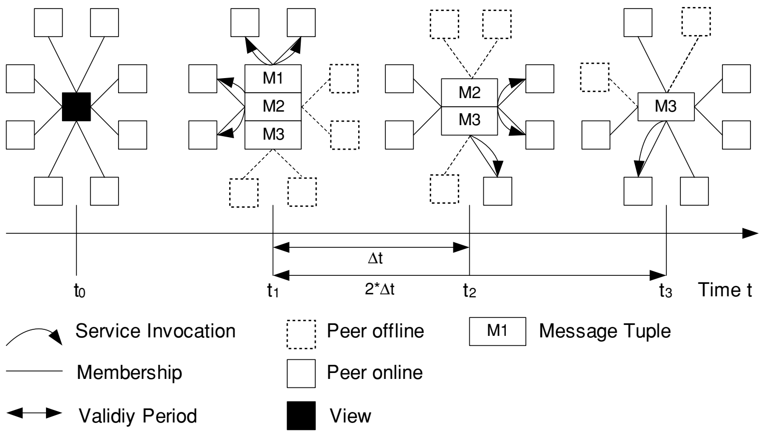


Fig. 11 VISR Web service invocation in decoupled operation mode

validity period expires. The tuple space also contains messages from peers to absent peers. This type of asynchronous messaging decouples the sender from the receiver completely. Figure 11 shows three examples of broadcast messages in disconnected operation. At time t_0 a view is created and contains a list of active peers. At time t_1 three different Web service requests are executed, denoted as tuples M1, M2 and M3. The first request at t_1 demands immediate response, thus synchronizing the sender and the receiver. The requestor places a message tuple (M1) into the tuple space and waits for the response of the Web service providers. In the example below, four peers are online at t_1 and their Web services are invoked. The second request (M2) provides a validity period of Δ and invokes Web services at t_1 and t_2 before the message tuple is being removed from the tuple space. The third request (M3) provides a validity period of 2*Δ and the Web services are invoked at t_1, t_2 and t_3 before the message tuple is removed. Note, that in our example only the third Web service request with the validity of 2*Δ actually provides the results of all peers of the view.

Synchronous messaging. In this mode, VISR peers use the SOAP interface for Web service invocation. In contrast to the decoupled operation mode, the consignee must be online to receive a message and the Web service requestor must know the peer addresses of the Web service provider. Furthermore, in contrast to the decoupled operation, VISR peers can select the peer that executes the service directly.

View dependent Web service invocation. The view dependent Web service invocation puts a Web service request into the context of a VISR view. Since VISR Views define interfaces of the provisioned Web services, Web service requestors do not need to be aware of the actual Web service binding and send Web service invocation messages with the parameters to Web service provider. Every VISR Web service provider is responsible for the correct transformation of the request to the actual Web service. Figure 12 illustrates the steps that are necessary for a view based invocation of a Web service.

5. VISR Web service discovery

This section discusses VISR's Web service discovery model. VISR provides two several types of interfaces for the matching of VISR service profiles. While supporting keyword-based

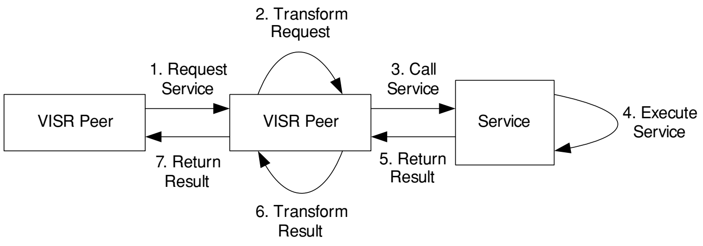


Fig. 12 . VISR context based Web service invocation

match of Web service descriptions, VISR improves upon this by employing the structure of VISR service profiles as potential matching criteria (see Section 5.1 for a detailed description of the matching algorithm). Furthermore, VISR supports XPath [13] expressions (Section 5.2) for the selection of desired Web services. Table 1 gives an overview of the different matching methods.

5.1. VISR structural matching algorithm

VISR's structural matching algorithm is based on binary string representations of XML documents. The algorithm operates tag name independent, i.e., tag names are not taken into account. The result of the transformation is a binary value (SSMA value). This value provides the means for the creation of a Web service metric by assigning a value to a Web service description. In addition, this value can be used to limit the actual search space, because structural compatible Web services provide at most an equal SSMA value.

The structural matching algorithm guarantees to find similar documents that contain at least one structural identical (sub) tree structure or a structure that can be transformed into the search structure. We illustrate the algorithm using a VISR service profile that provides a list of VISR peers that are members of a view. Below is the example of a VISR service profile that returns the peer list of a VISR view. The corresponding tree structure is depicted in Fig. 13.

The algorithm builds string representations of the trees that are to be compared. This is done level by level using a breadth first algorithm. Every level, respectively every node of the tree, is transformed into a string that contains the letter "1" for a parent node and the

Table 1 Matching types of VISR

Type	Description
Keyword match	Searches VISR profiles for the occurrence of arbitrary keywords
XPath match	Searches for VISR profiles by the specification of XPath expressions
Structural match	Selects VISR profiles using VISR's structural matching algorithm
Structural and keyword match	Selects VISR profiles using a combination of keywords and structural information

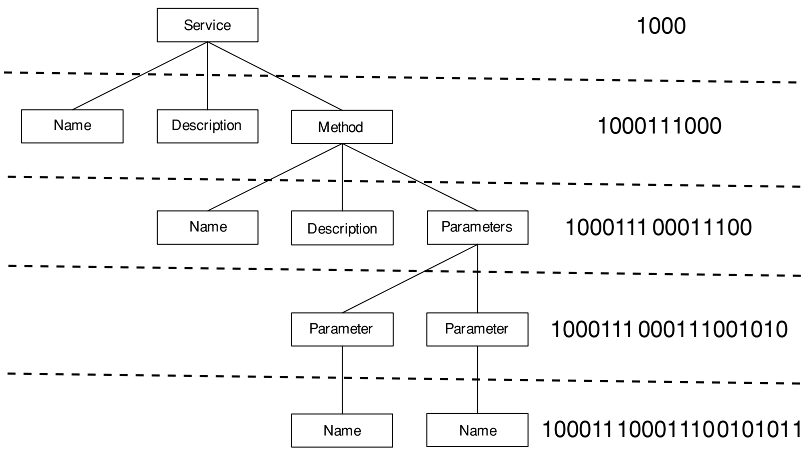


Fig. 13 Transformation of VISR service profiles into String representation. On the right hand side the transformation of the current level of the tree is depicted. The lowest level contains the actual SSMA value of the Web service description

letter "0" for a child. In the example above, the resulting substrings are: [1000], [1], [1], [1000], [1], [1], [100], [10], [10], [1] and [1]. The corresponding value of the Web service description is 1000111000111001010111.

The comparison starts with the first substring of the source tree (= tree structure we are looking for) and searches the string representation of the target tree (= tree structure of the searched tree) for the first occurrence of the substring. If a corresponding string is found then the next substring of the source tree is taken as input and the search resumes at the following substring of the target tree. If the end of the target string is reached, without a hit then the target contains no corresponding tree structure. Otherwise, the target tree contains a similar tree structure that can be matched on the target tree (see Fig. 14).

5.2. XPath based discovery of VISR service profiles

The XPath based discovery uses XPath expressions for the discovery of VISR service profiles. An example of such an expression that selects VISR service profiles that offer two parameters is given below.

```
//*[count(Parameters) = 2]
```

6. Application scenario

We introduced an application scenario concerning a film production company in our previous work [26]. This section describes a concrete application scenario for our prototype. We consider a film production company that produces a movie. The film production company must coordinate the activities needed for the shooting of a movie. The coordination is distributed among several working teams, which in turn must coordinate their activities.

Our working example consists of two different services. We consider a service that provides the current activity of a film crew member and a service that notifies film crew members about arbitrary events, for example about meetings, etc. Code snippets of the

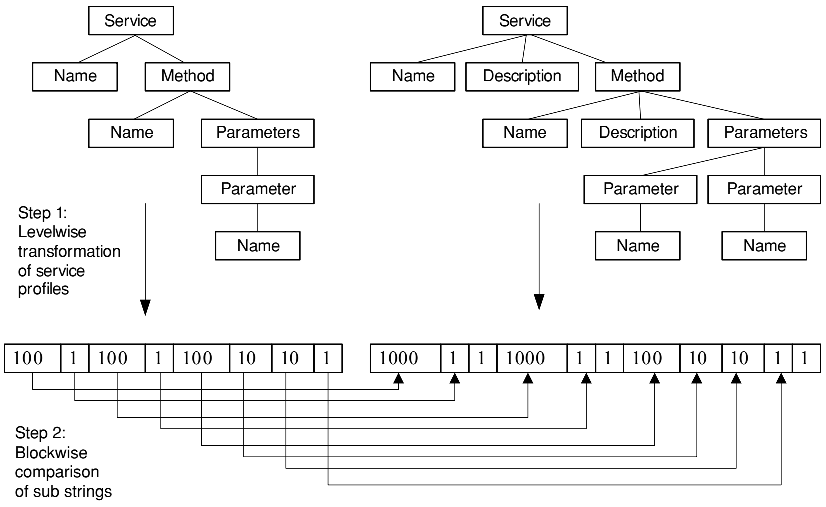


Fig. 14 Overview of SSMA

services are illustrated below. For the sake of brevity, we omitted the usage part of the VISR service profiles.

```
  <Service>
    <Name>Get current activity</Name>
    <Description>Returns the current activity of the film crew member
    </Description>
    <Method>
      <Name>getCurrentActivity</Name>
        </Input>
        <Output>
         <Attribute>Activity</Attribute>
        </Output>
    </Method>
    ..
  </Service>
  <Service>
    <Name>Notify</Name>
    <Description>Notifies the film crew member about the occurrence of
  an arbitrary event.
    </Description>
    <Method>
      <Name>notify</Name>
        <Input>
         <Parameter>message</Parameter>
        </Input>
        </Output>
    </Method>
    ..
  </Service>
```

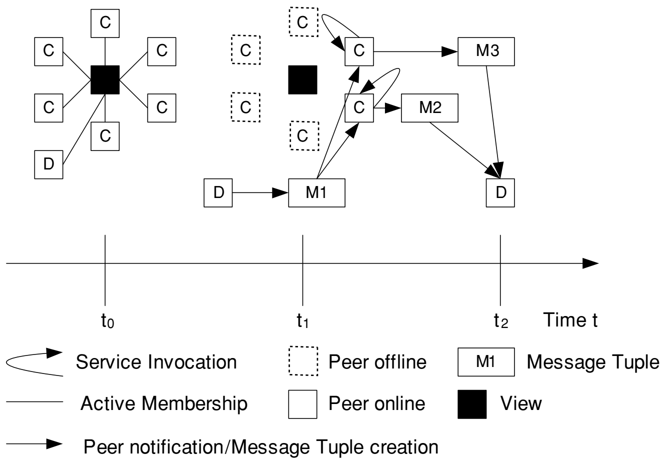


Fig. 15 Web service invocation of VISR using tuples for communication

We now assume that our film crews are not permanently online and that the service provision is offered only temporarily. Hence the associated Web services registry entries do not exist permanently. Figure 15 shows the different states of the distributed Web service registry, concerning the available number of VISR peers. At time t_0 every member of the film crew (C) and the film director (D) are online and provide the specified services. At time t_1, only two out of six members of the film crew are online. We now assume, that the film crew director invokes the Web service called "get current activity" by putting a message tuple M1 into the tuple space at time t_1. At that time, two peers are online and are notified by the tuple space infrastructure that a new message tuple is available. The peers invoke their services and write the result info the tuple space (message tuples M2 and M3). The film director polls the tuple space at time t_2 and receives the results of the service invocations.

7. Related work

The project presented in [20] presents an extension to the Service Oriented Architecture and introduces the notion of channels that can be compared to the VISR view concept. Channels serve as means to structure a global tuple space for coordination purposes. Our approach uses VISR views to structure a global tuple space but in contrast with the work in [20], VISR views provide a common interface description for Web services.

EgoSpaces [22] provides an abstraction called view that exposes some similarity to VISR views. Views in EgoSpaces provide scalable coordination in an ad hoc network. In EgoSpaces, an agent sees the world through a set of personalized views. In contrast to EgoSpaces, VISR does not provide an agent based solution but VISR views also allow personalized information.

The work in [23] employs a federation of UDDI-enabled peer registries. These registries operate in a decentralized fashion and provide federations of peers with related or similar services. These federations, or peer group syndications, are administered by a super peer that manages the publishing, joining/leaving and service subscriptions. This concept is similar to VISR community concept, based on VISR views, but needs super peers to operate properly.

VISR also introduces the view originator that serves as administrator for views, but does not provide additional meta data services like management of login/logout of other peers.

The WSDA [18] grid architecture provides a semitransparent umbrella for distributed data. WSDA does not focus explicitly on Web service registry but provides discovery functions for distributed information. WSDA uses a tuple space model to store information among nodes of the network. In comparison with VISR, WSDA registry information can be of any format, WSDA only provides data tuples that are capable to store Web service descriptions. Tuples are used to represent VISR service profiles of transient clients. In comparison with VISR, WSDA does not consider Web service communities or different Web service registry implementations.

VISR encompasses contributions of the SELF-SERV [21] project. SELF-SERV exploits the concept of communities. Communities offer a well defined class of services with common capabilities. A community delegates the execution of a service to a member according to a selection policy. VISR follows a similar idea, regarding the structuring of communities, respectively views, but VISR does not provide dynamic provider selection as SELF-SERV does.

VISR proposes a community concept similar to the WebBis [9] communities. WebBis offers two types of communities (push and pull). These communities correspond roughly to dynamic respective static views of VISR. In comparison, VISR focuses on the actual data models of registries and their declarative integration. WebBis follows a more abstract approach by using an ontological based approach to structure push respectively pull communities. In addition, WebBis proposes service wrappers that are used to provide a common service description. VISR also provides common Web service descriptions that are specified by VISR service profiles. VISR service profiles expose some similarities to WebBis service wrappers but operate on a different level. VISR service profiles offer a common declarative service description whereas WebBis offers an object oriented approach. WebBis also handles Web service composition and provides an event handling system to monitor changes. Web service and composition and change monitoring are not covered by our work.

The community concept described in [35] is similar to the concept of VISR communities. The authors introduce the generic operations for communities that are similar to the abstract interface descriptions in VISR. In contrast to VISR, the community concept is based on a meta data ontology for communities. Furthermore, the communities are published in UDDI registries, whereas VISR offers a tuple space for the persistence of community information.

The ebXML [2–5] standard introduces the concept of Web service registry federation. Federated Web service registries form loosely coupled unions of related Web services. These federations appear as a single logical registry to clients. This approach shares some of the objectives of our work. VISR views are similar to federations whereby ebXML federations focus on the lifecycle of registry objects and replication issues. Our work focuses on the creation of long lasting static views and dynamic views. Dynamic views, i.e., ad hoc federations are not covered by ebXML federations. ebXML federations also do not cover the issue of transient registry entries provided by transient Web service providers. VISR offers lightweight clients to present a fully distributed non replicated Web service registry without the need of additional administrative overhead.

The UDDI [1] standard also acknowledges the need for a distributed registry structure. It introduces replication among distributed registries but focuses mainly on the actual distribution of registry entries. In favor of a flexible integration of lightweight VISR peers VISR does not include a replication model, because replication increases the administrative overhead. UDDI allows the creation of private registries that are physically separated from other registries. In comparison, VISR offers a more flexible approach. VISR uses views to create logical separations of registry data and controls registry access with the help of a membership service.

VISR utilizes the concept of tracker sites, similar to the Bittorrent [25] network. Trackers coordinate activities between peers of the Bittorrent network. They can be regarded as communication hubs that provide a distributed registry. VISR takes a step further, since the creation of trackers is comparable with the creation of VISR views. In contrast, VISR offers a global querying system, whereas the Bittorrent limits querying only to tracker sites.

Related work concerning the evaluation of structural similarities between XML documents can be found in [29–31]. The basic idea of these algorithms is the calculation of the (minimum) edit distance between string representations of different DOM trees that reflect the hierarchical structure of XML documents. A different approach is taken by Flesca et. al [32] who evaluated structural similarity between XML documents using a time series representation of XML documents. For the benefit of simplicity, our proposed simple structure matching algorithm (SSMA) follows a more naïve approach, because our algorithm does not calculate the actual edit distance between XML documents. Furthermore, our algorithm operates with documents that originate from the same schema definition and therefore provide a very helpful limitation regarding potential differences between compared XML documents.

In the Web service area, there are several research projects that focus on Web service context. VISR encompasses contributions from several sources. Mostéfaoui and Mostefaoui [28] present an agent-based architecture that provides service selection based on a rating system for Web services. The work in [34] describes a framework that models user tasks as coalitions of abstract services. Keidl and Kemper [37] use the UDDI data model to provide context information about Web services. Doulkeridis et al. [7] provides a context model for mobile services based on a graph representation of context.

8. Conclusion and future work

In this paper, we presented a framework for a distributed virtual Web service registry in a dynamic environment based on tuple spaces. Tuple spaces serve as primary communication means and as storage media. We think of tuple spaces as a cornerstone for dynamic Web service provisioning because of their device independency and decoupled communication capabilities.

Our approach also considers a peer to peer network as a feasible solution for the integration of transient Web service providers. Transient Web service providers join and leave the VISR peer to peer network dynamically. Usually, they are devices that offer limited processing power and memory capacity. To facilitate the integration of transient Web service provider we presented the VISR view concept. Views are abstract contexts in which Web services are published and can be regarded as virtual registries. Views also allow the creation of Web service communities that provide related Web services with equal service invocation patterns. In our approach, views furthermore include context information that is out of the scope of common Web service registry descriptions.

A significant part of the paper discusses the matching algorithm of VISR. We make use of structural information of VISR service profiles to support the discovery of Web services. Our next step is the extension of our algorithm with the help of vector space oriented methods [8].

Due to lack of availability of distributed tuple space implementations, our current proof of concept prototype uses a centralized version of tuple spaces. We intend to move to fully distributed tuple spaces in our future work. We think that with the advent of distributed tuple spaces further enhancements in flexibility can be achieved. We currently investigate two possible scenarios for the next implementation of the distributed tuple space. We examine a distributed tuple space solution based on the P-Grid [36] infrastructure that allows a scaleable distribution of tuples. We furthermore consider the possibility of moving to version 3 of IBM's tuple space implementation.

Appendix

This section provides the schema files and corresponding APIs of VISR views, VISR service and VISR peer profiles.

VISR Service profile

```
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="Service">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Description"/>
    <xs:element name="Method" type="MethodType"
maxOccurs="unbounded"/>
    <xs:element name="Usage">
     <xs:complexType>
      <xs:sequence>
        <xs:element name="Description"/>
        <xs:element name="Example" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Invocation"
maxOccurs="unbounded">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="Method"
type="MethodType"/>
                <xs:element name="Result">
                 <xs:complexType>
                  <xs:sequence maxOccurs="unbounded">
                   <xs:element name="Attribute"
type="AttributeType"/>
                   </xs:sequence>
                  </xs:complexType>
                 </xs:element>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
```

```
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:complexType name="AttributeType">
        <xs:sequence>
         <xs:sequence>
          <xs:element name="Name"/>
          <xs:element name="Value"/>
         </xs:sequence>
         <xs:element name="Attribute" type="AttributeType" minOccurs="0"
  maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
       <xs:complexType name="ParameterType">
        <xs:sequence>
         <xs:element name="Name"/>
         <xs:element name="Value" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" use="optional"/>
       </xs:complexType>
       <xs:complexType name="MethodType">
        <xs:sequence>
         <xs:element name="Name"/>
         <xs:element name="Description"/>
         <xs:element name="Parameters">
          <xs:complexType>
           <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Parameter" type="ParameterType"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:schema>
```

VISR Service profile API

```
  public class VISRServiceProfile extends VISRProfile {
    VISRServiceProfile();
    public String getID();
    public String getDescription(String description);
    public void setDescription();
    public void setName(Stirng name);
    public String getName();
    public void addAttribute(Attribute attribute);
    public AttributeList getAttributeList();
    public void removeAttribute(Attribute attribute);
    public void addMethod(Method method);
    public void removeMethod(Method method);
    public MethodList getMethodList();
    public void setUsage(Usage usage);
    public Usage getUsage();
    public VISRServiceList matchService(Query query);
    public VISRServiceList matchService(VISRViewProfile view);
  }
```

VISR View profile

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
     <xs:element name="View">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="ID"/>
        <xs:element name="Name"/>
        <xs:element name="Description"/>
        <xs:element name="Filter">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="Input">
            <xs:complexType>
             <xs:sequence maxOccurs="unbounded">
              <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Attribute" type="AttributeType"/>
              </xs:sequence>
              <xs:sequence minOccurs="0" maxOccurs="unbounded">
               <xs:element name="Method" type="MethodType"/>
              </xs:sequence>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="Output">
            <xs:complexType>
             <xs:sequence maxOccurs="unbounded">
              <xs:sequence minOccurs="0" maxOccurs="unbounded">
               <xs:element name="Attribute" type="AttributeType"/>
              </xs:sequence>
              <xs:sequence minOccurs="0" maxOccurs="unbounded">
               <xs:element name="Method" type="MethodType"/>
              </xs:sequence>
             </xs:sequence>
            </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:complexType name="AttributeType">
    <xs:sequence>
     <xs:sequence>
      <xs:element name="Name"/>
      <xs:element name="Value"/>
     </xs:sequence>
     <xs:element name="Attribute" type="AttributeType" minOccurs="0"
         maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ParameterType">
    <xs:sequence>
     <xs:element name="Name"/>
     <xs:element name="Value" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="optional"/>
   </xs:complexType>
   <xs:complexType name="MethodType">
    <xs:sequence>
     <xs:element name="Name"/>
     <xs:element name="Description"/>
     <xs:element name="Parameters">
      <xs:complexType>
       <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Parameter" type="ParameterType"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
</xs:schema>
```

VISR View API

```
public class VISRViewProfile extends VISRProfile {
  VISRViewProfile();
  public String getID();
  public String getDescription(String description);
  public void setDescription();
  public void setName(Stirng name);
  public String getName();
```

```
    public int getServicePort(int servicePort);
    public void setServicePort();
    public void setInputFilter(InputFilter inputFilter);
    public InputFilter getInputFilter();
    public void setOutputFilter(OutputFilter outputFilter);
    public OutputFilter getOutputFilter();
    public VISRPeerList getVISRPeerList();
    public void addVISRPeer(VISRPeerProfile VISRPeerProfile);
    public void removeVISRPeer(VISRPeerProfile VISRPeerProfile);
  }
```

VISR peer profile

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:element name="Peer">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID"/>
      <xs:element name="Name"/>
      <xs:element name="Description"/>
      <xs:element name="Type"/>
      <xs:element name="IP"/>
      <xs:element name="ServicePort"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
</xs:schema>
```

VISR peer profile API

```
public class VISRPeerProfile extends VISRProfile {
   VISRPeerProfile();
   public String getID();
   public String getDescription(String description);
   public void setDescription();
   public void setName(Stirng name);
   public String getName();
   public int getServicePort(int servicePort);
   public void setServicePort();
   public void setVISRPeerType(VISRPeerType VISRPeerType);
   public VISRPeerType getVISRPeerType();
   public ServiceList getServiceList();
   public void publishService();
   public Service findService();
   public void unpublishService(Service service);
   public void publishView(View view);
   public void unpublishView(View view);
 }
```

References

1. OASIS. Universal Description, Discovery and Integration: UDDI Technical White paper, 2000. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf. .
2. OASIS. OASIS/ebXML registry services Specification v2.5, 2003. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf.
3. OASIS. OASIS/ebXML Technical Architecture Specification, 2001. http://www.ebxml.org/specs/ebTA.pdf.
4. OASIS/ebXML registry Information Model v2.5, 2003. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrim-2.5.pdf.
5. OASIS. Business Process Specification Schema, 2001. http://www.ebxml.org/specs/ebBPSS.pdf.
6. R. Tolksdorf, and D. Glaubitz, "Coordinating web-based systems with documents in XMLSpaces," CoopIS 2001, LNCS 2172, Springer-Verlag, Berlin Heidelberg, 2001, pp. 356–370.
7. C. Doulkeridis, E. Valavanis, and M. Vazirgiannis, "Towards a context-aware service directory," in B. Benatallah and M.-C. Shan (Eds.), TES 2003, LNCS 2819, Springer-Verlag, Berlin Heidelberg, 2003, pp. 54–65.
8. S. Dustdar and C. Platzer, "A vector space search engine for Web services," in IEEE European Conference on Web services (ECOWS), IEEE Computer Society Press, 14–16 November 2005.
9. B. Medjahed, B. Benatallah, A. Bouguettaya, and A. Elmagarmid, "WebBIS: An infrastructure for agile integration of web services," International Journal of Cooperative Information Systems, vol. 13, no. 2, pp. 121–158, 2004.
10. W3C. Web Services Architecture W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/wsa.pdf.
11. IBM. TSpaces, 2003. http://www.alphaworks.ibm.com/tech/tspaces.
12. W3C. XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.
13. W3C. XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999. http://w3c.org/TR/xpath.
14. W3C. SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation 24 June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.
15. W3C. SOAP Version 1.2 Part 2: Adjuncts W3C Recommendation 24 June 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624/.
16. W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language W3C Working Draft 3 August 2004. http://www.w3.org/TR/2004/WD-wsdl20-20040803.
17. W3C. Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions W3C Working Draft 3 August 2004. http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803.
18. W3C. Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings W3C Working Draft 3 August 2004. http://www.w3.org/TR/2004/WD-wsdl20-bindings-20040803.
19. D. Gelernter, "Generative communication in Linda, ACM Transactions on Programming Languages and Systems," vol. 7, no. 1, pp. 80–112, 1985.
20. P. Alvarez, J.A. Banares, and P.R. Muro-Medrano, "An architectural pattern to extend the interaction model between web-services: The location-based service context," ICSOC 2003, LNCS 2910, Springer-Verlag Berlin Heidelberg, 2003, pp. 271–286.
21. B. Benatallah, M. Dumas, and Q.Z. Sheng, "Faciliating the rapid development and scalable orchestration of composite web services," Distributed and Parallel Databases, vol. 17, pp. 5–37, 2005.
22. C. Julien, G.C. Roman, and J. Payton, "Bringing context-awareness to applications in Ad hoc mobile networks," Technical Report WUCSE-04-18, Washington University, Department of Computer Science and Engineering, St. Louis, 2004.
23. M.P. Papazoglou, B.J. Krämer, and J. Yang, "Leveraging Web-services and Peer-to-Peer networks," CAiSE 2003, LNCS 2681, 2003, pp. 485–501.
24. SUN, 2004. Java 1.5. java.sun.com.
25. B. Cohen, 2003. "Incentives build robustness in BitTorrent," www.bittorrent.com.
26. S. Dustdar, M. Treiber, "A view based analysis on Web service registries," Distributed and Parallel Databases, forthcoming.
27. A. Tsalgatidou and T. Pilioura, "An overview of standards and related technology in Web services," Distributed and Parallel Databases, vol. 12, pp. 135–162, 2002. Kluwer Academic Publishers.
28. K. Mostéfaoui and G.K. Mostefaoui, "Towards a contextualisation of service discovery and composition for pervasive environments," in Proc. of the Workshop on Web-services and Agent-based Engineering (WSABE), 2003.

29. E. Bertinoa, G. Guerrini, and M. Mesitia, "A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications," Information Systems, vol. 29, no. 1, 2004.
30. Y. Wang, D.J. DeWitt, and J.Y. Cai, "X-Diff: An effective change detection algorithm for XML documents," Proc. Int'l Conf. Data Eng. (ICDE '03), 2003, pp. 519–530.
31. A. Nierman and H.V. Jagadish, "Evaluating structural similarity in XML documents," in Proc. Int'l Workshop Web and Databases (WebDB '02), 2002.
32. S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese, "Fast detection of XML structural similarity," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 2, 2005.
33. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, A System of Patterns, Wiley, 1996.
34. J.P. Sousa and D. Garlan, "Aura: an Architectural Framework for user mobility in ubiquitous computing environments," in Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA), 2002, pp. 29–43.
35. B. Medjahed and A. Bouguettaya, "A dynamic foundational architecture for semantic Web services," Distributed and Parallel Databases, vol. 17, pp. 179–206, 2005. Springer-Verlag, Berlin Heidelberg.
36. K. Aberer, A. Datta, and M. Hauswirth, "P-Grid: Dynamics of self-organization processes in structured P2P systems," Peer-to-Peer Systems and Applications, Lecture Notes in Computer Science, LNCS 3845, Springer-Verlag Berlin Heidelberg, 2005.
37. M. Keidl and A. Kemper, "Towards context-aware adaptable Web services," EDBT 2004, LNCS 2992, Springer-Verlag Berlin Heidelberg, 2004, pp. 826–829.