View based Integration of heterogeneous Web service Registries – the case of VISR

Schahram Dustdar and Martin Treiber

Distributed Systems Group
Institute of Information Systems
Vienna University of Technology
{dustdar@infosys.tuwien.ac.at|m.teiber@infosys.tuwien.ac.at}

Abstract. Despite all standardization efforts in the Web service area, several different incompatible Web service registry implementations exist. The initial focus of these implementations was geared towards working with a centralized Universal Business Registry (UBR). However, these centralized approaches tend to be bottlenecks regarding performance and fault tolerance. A proposed solution is the replication of registry information among multiple distributed Web service registries. In addition, the creation of specialized Web service registries leads to a large number of different Web service registries. This leads to a situation where the search for a particular Web service becomes a very complex task. Besides, Web service provisioning includes a considerable administrative overhead when dealing with transient Web services. Transient Web services exist only for a limited lifetime and in a certain context. In this paper, we propose the VISR (**V**iew based **I**ntegration of Web **S**ervice **R**egistries) peer to peer architecture for the transparent integration of multiple Web service registries and transient Web service providers. This work focuses on the integration concept of multiple Web service registries and transient Web service providers. The integration concept relies on so-called views. Views provide the needed abstractions for the seamless integration on the different registries. Views use common lightweight Web service profiles that serve as unified global data model. VISR Web service profiles allow the flexible extension of registry entries with value added information without changing the original Web service registry entries. To illustrate the view concept, we introduce a simple grammar (View Description Language) for view descriptions that is used in the working example throughout the paper. We present Web service communities as a possible application of the view concept and show how different types of Web services providers respectively their registries are integrated into a unified global data model.

Keywords: Web service Discovery, Web service Registry Integration, Distributed Web service Registry

1 Introduction

Current Web service registries [24] like UDDI [1, 19] and ebXML [2, 3, 4] provide no means for the integration of different Web service registries. UDDI and ebXML registries have similar intensions, i.e., the publishing and discovery of Web services, but their data models differ and they provide different mechanism for the discovery respectively publishing of Web services. Especially ebXML offers - in comparison to UDDI - a much broader approach in terms of Web service description since ebXML registries support business coordination protocols [5, 6]. These Web service registry implementations usually follow a centralized approach, with a single central Universal Business Registry (UBR). As the number of services grows, a single UBR might prove as a bottleneck. To avoid this potential bottleneck UBR registries are distributed among several servers. To ensure that every registry provides the same registry information, these registries are synchronized periodically and provide several distributed access points.

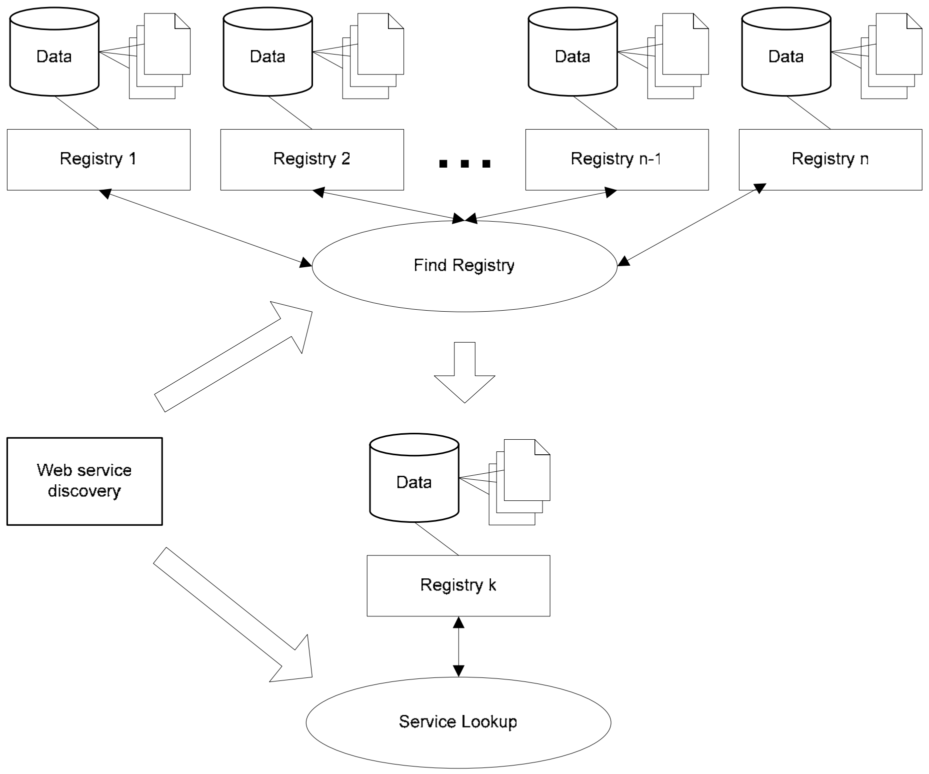

Fig. 1. Web service discovery in distributed registries

However, as Web services become more dynamic, consistency between distributed registries cannot be guaranteed any more. We consider Web services that

Consider a transient Web service `S` that registers itself in a registry `A`. If this registry is replicated before the transient Web service `S` is deleted, every replication copy of registry `A` contains the registry entry of the Web service `S`. After Web service `S` is deleted, every replication copy contains a registry entry that is inconsistent with the original registry until another replication round has taken place. In addition, a trend to Web service communities [9] can be expected. Web service communities are groups of related Web services that are published in a single Web service registry and share common interests.

We presented an application scenario concerning a movie production company in our previous work [22]. We now focus on the coordination activities of a movie director, which can roughly divided into two phases, (i) the project planning phase and (ii) the coordination of the movie teams during the actual project. During the first phase a movie director needs information about available services of the different movie crews, for example about stuntmen crews, makeup artists, etc. In order to find the services, the movie director must browse through available registries to find adequate movie teams that provide the needed services (see Figure 1). Because of potential different registry implementations, this activity involves considerable overhead and needs different registry browser implementations.

During the second phase, the execution phase, a movie director must coordinate the activities of the different movie crews. The coordination effort needs the support of different registries in order to restructure Web service registry content into Web service communities. For example, it may be necessary to organize a Web service community that provides services for controlling costs of certain activities (charting of trucks for transport of equipment, etc.). Therefore, every movie crew must provide a cost control Web service that is a member of the cost management community. It may also be necessary to reorganize existing communities for a short time, for instance during joint meetings of film crews. During the meeting it may be necessary to share data (presentations, pictures, documents, etc.). To accomplish this, a file sharing Web service may be offered where community members can share their files.

Another example that illustrates the problems of distributed heterogeneous Web service registries are companies that offer business information (company reporting, company ratings, etc.) about companies in different countries. These companies provide (private) Web service registries where they register their business information services. So if a Web service requestor wants to search for business information about companies of different countries, a search pattern as depicted in Figure 1 emerges – leading to several lookups in different often heterogeneous Web service registries.

To avoid these limitations, we propose a distributed Web service registry solution. The VISR (**Vi**ew based Web **s**ervice **r**egistry integration) architecture provides a peer to peer network of interconnected Web service registries and transient Web service providers (see Figure 2). The VISR peer to peer architecture offers a meta data model that provides an integration concept that unifies these different Web service registries and transient Web service providers in a transparent manner. The integration concept (regarded as view concept in the rest of the paper) supports the integration of registries and transient Web service providers. It includes an extension mechanism for future extensions regarding the registry data model. The view concept also provides the means for the building of Web service communities.

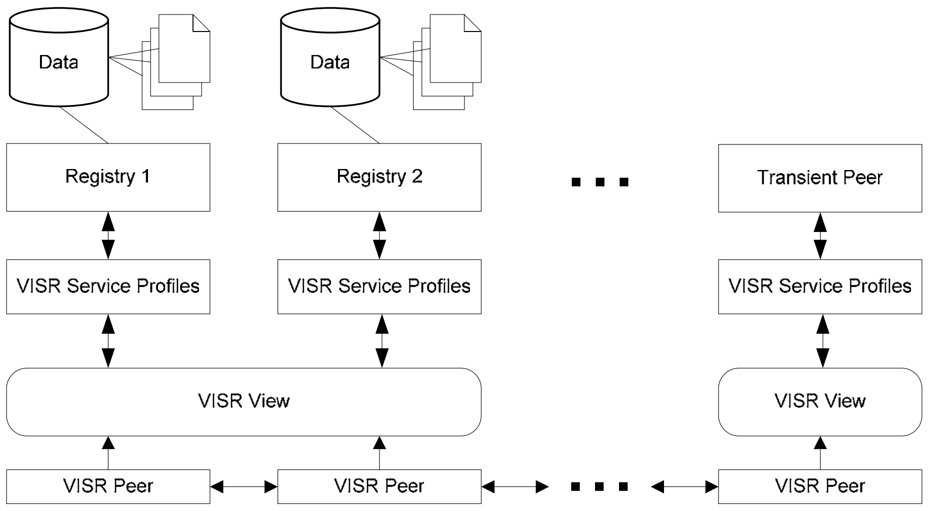

Fig. 2. VISR Overview

The remainder of the paper is organized as follows. Section 2 discusses the VISR View concept in detail. Section 3 presents the VISR peer to peer architecture. Section 4 presents related work. Section 5 concludes the paper.

2 VISR View Concept

One of the main goals of our work is to create an abstract context that allows the integration of heterogeneous Web service registries and transient Web service provider into a common distributed virtual Web service registry [21]. Our approach uses a peer to peer architecture that provides for unified Web service registry access (see Section 4 for a detailed discussion of the VISR architecture) of distributed Web service registries.

We consider VISR views as cornerstone for the integration concept. VISR views specify how Web services are published, discovered and invoked by peers of the VISR peer to peer network. In particular, the VISR view concept addresses the following issues:

- **Web service communities.** VISR views provide the means for a logical integration of heterogeneous Web Service registries and transient Web service providers.

- **Additional meta data.** VISR views add meta information to existing Web service registries without changing the data model of the existing Web service registry.

- **Common Web service definition.** VISR views provide a common abstract lightweight Web service definition (VISR service profiles) in order to integrate heterogeneous Web service descriptions and to provide the means for the integration of transient Web service providers.

- **Common Web service invocation.** Along with a common Web service definition, VISR views provide the means for unified Web service invocation using views as abstraction layer.

2.1 VISR communities

The view concept allows the definition of Web service communities [9] that integrate similar Web service registry entries to a community. A VISR community is essentially a group of Web services that share common interests. VISR views provide abstract descriptions, i.e., the view description and offer identical service interfaces. This allows an abstract specification of services without regarding to the actual Web service provider and the actual Web service. In order to become a community member, the Web service provider must implement these interfaces with the help of local mappings and filters.

Consider the working example with a community concerning the movie project, defined by the director of the movie. A Web service provider that publishes community adequate Web services does not need to enrich its Web service description with additional context information about the movie project. The Web service provider can rely on context information provided by the community, in this case on information provided by the movie director. This enables Web services to be part of several views at the same time without the need of any change in the original Web service description.

Another aspect of VISR communities concerns the peer to peer communication. Every VISR community defines a logical communication space that is available exclusively to community members. The communication space is based on the Blackboard architectural pattern that provides concepts for decoupled peer to peer communication. This pattern enables the implicit integration of transient Web service providers, since the communication between transient peers is decoupled in space and time.

Our working example illustrates this behavior regarding decoupled message exchange between members of nimble project teams, like for example makeup artist teams. These teams, respectively their members, operate all-over the movie set. This may lead to frequent disconnections from the communication network. A dedicated (transient) VISR makeup artist community can provide a communication space that supports decoupled communication among its members based on message tuples.

2.1.1 Transient vs. Stable Communities

VISR distinguishes between two types of communities: transient communities and stable communities. We discuss the different communities in detail in the following sections.

Every view description defines a Web service community. Depending on the type of the view, a view can either be of (1) transient or (2) stable character. A VISR peer can join and leave a community at any time. When joining a VISR community respectively a view a VISR peer must provide Web services that meet the requirements of the community. When a community defines certain interfaces and data fields then the VISR peer must assure that these interfaces are also available at the peer.

Transient communities. A transient community (view) is a loosely coupled group of Web services providers that temporarily meet the requirements of a view description and may exist only for a certain time. In order to join a transient community, a Web service provider must contact the community creator and provide the Web services defined by the community. The community creator then stores a reference to the peer in a local peer list. The peer list is used for peer to peer communication among community members. A transient Web service provider may also join and leave a transient community at arbitrary times and later return to the community, provided that during the offline time the community creator is still online (see Figure 3). Transient communities only exist as long as there are members online that provide the specified services.

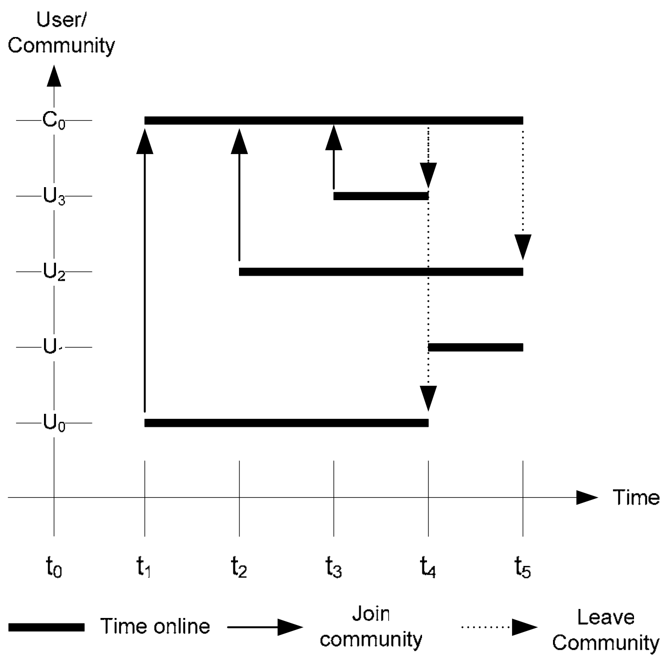

Fig. 3. Timelines of VISR peers and a VISR community. At time t_1 a community c_0 is created and joined by user u_0. At t_2 user u_2 joins the community c_0. At t_3 user u_3 joins the community c_0. At t_4 user u_0 (the view creator) and user u_3 leave the community c_0. At t_4 user u_1 tries to join the community c_0 but since that it is not possible without the view creator online the peer cannot join the community c_0. At t_5 user u_2 leaves the network and the community c_0 ceases to exist.

Transient community example. Consider the working example with extras that are needed for a mass scene. The movie director defines an adequate view/community and publishes it. Every member of the movie crew who is able to provide the service can join the view at any time before the actual shooting. After the scene is completed the view is implicitly invalidated either by a temporal or a prior defined event. Thus, a transient community provides the necessary features for the building of ad-hoc communities. The membership is specified through the properties “member” and “salary”.

Stable communities. In contrast to transient communities, a stable Web service community is a long-term relationship among related Web services respectively Web service providers. A stable community exists either for a previously defined amount of time (see Figure 4). A stable view is also replicated among its members. In contrast to transient communities, a peer that wishes to join the community does not need to contact the community originator, it is sufficient to contact an arbitrary member of the community to join the community. A stable membership is defined by the explicit termination, i.e., the creator of the community removes the community. Every registered peer is removed from the peer list and every peer is notified by the occurrence of this event. Registered peers remove their view description replica.

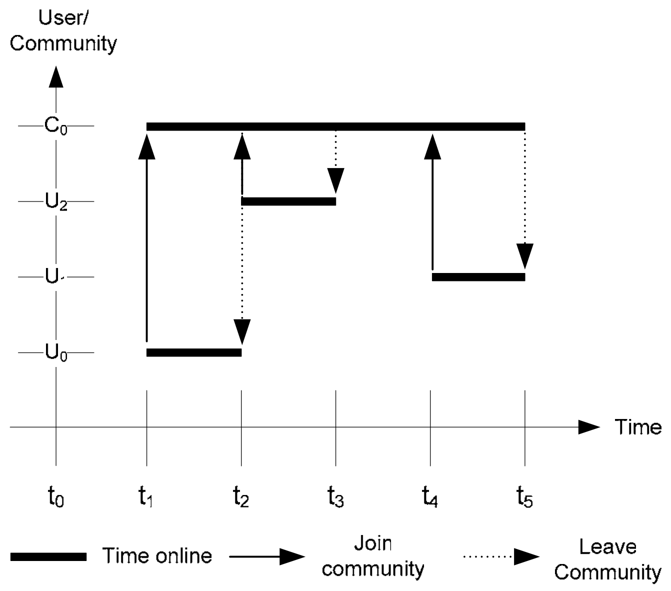

Fig. 4. Timelines of VISR peers and a stable VISR community. At time t_1 a community c_0 is created and joined by user u_0 with a pre-defined lifespan $\Delta t=4$. At t_2 user u_2 joins the community c_0 and user u_0, the view creator leaves the network. At t_3 user u_2 leaves the community c_0, leaving an empty community c_0 behind. At t_4 user u_1 joins the community c_0. At t_5 user u_1 leaves the community c_0 and c_0 is removed from the network.

Stable Community Example. Consider our working example with the relationship between the movie director and his assistance team. The assistance team works throughout the movie project for the director. If the movie project ends the assistance team member receive a notification and leave the community. Note, that in contrast to transient communities, stable communities also exist if no member is online, because every member holds a replication of the community description.

2.2 VISR View Profiles

VISR view profiles represent VISR views. The main objective of VISR view profiles is to structure Web service registry data in a logical manner without regard to the actual physical distribution of Web service registry entries. VISR view profiles include descriptions of the functionality of sets of Web services in an arbitrary context. VISR view profiles model meta data about views. In addition, VISR views provide external visible abstract descriptions of Web services. These descriptions are interface specifications.

From a logical point of view, VISR View profiles consist of three parts (see Figure 5). Views provide two abstract interface specifications (Input and Output filter) and a mapping component (View-Registry Mapping).

The interface specification consists of the input and output filter. The input filter has two purposes. The input filter receives all incoming requests and transforms the requests according to the view specification. In addition input filter allow the definition of search criteria that pre select all underlying registry entries. The corresponding counterpart of the input filter is the output filter. The output filter specifies the externally visible method signatures and data fields. The actual mapping of these interfaces to the underlying registry data model is specified by the mapping component. Mappings are declared with the help of rules that define which elements of the registry data model are mapped onto the view elements. The results of the transformation process are VISR service profiles that provide registry information.

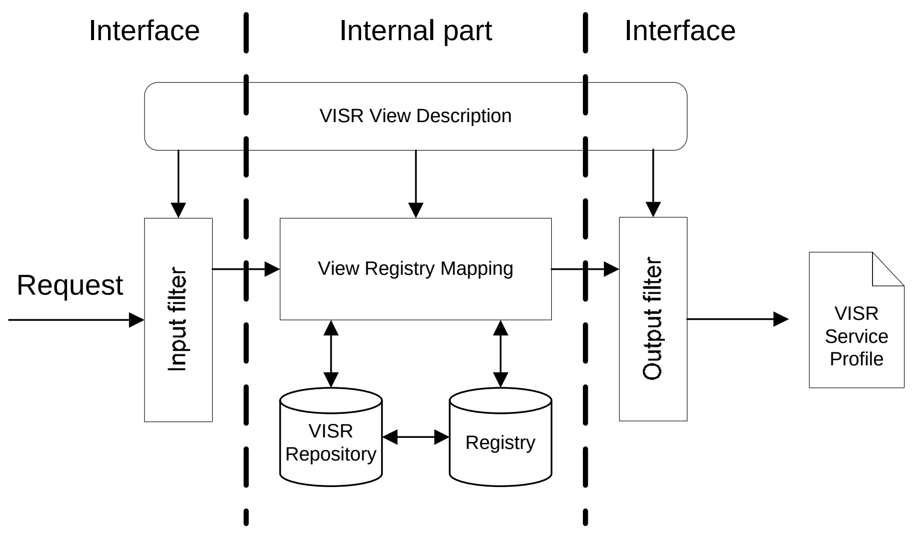

Fig. 5. VISR View overview

These three logical categories are modeled into four different classes:

View class. The `View` class encapsulates general information about the view. It consists of seven attributes that provide information about the view. It includes a `Creator` attribute to identify the creator of the view. The objective of the creator attribute is to define a VISR peer that is responsible for the administration of the view. The Originator maintains lists of all peers that are members of the view, to propagate messages efficiently. Table 1 summarizes all available fields of the view class.

Property	Description
Name	A brief, human readable name of the view
ID	The identifier of the view
CreateDate	Date, when the view was created
ExpireDate	Date, when the view expires
Type	Type of the view (transient or stable)
Originator	Creator of the view
Description	A brief, human readable description of the view.

Table 1. Available fields in the View class

Filter class. The `Filter` class provides interfaces for the input and output filter of views. The output filer describes the "public face" of views, or, in other words, what functionality view conform Web services provide. The input filter is the logical counterpart of output filters. It transforms incoming requests according to view descriptions, and forwards the result of these transformations to the registry. Table 2 summarizes the data format of filters.

Property	Description
Description	A brief, human readable name of the filter
Type	Type of filter (input or output)
Expression	XSL Transformation

Table 2. Available fields in the Filter class

VISR filter rules are implemented using XSLT expressions. If the registry provides matching entries, a set of matching entities is returned, otherwise an empty set is returned. To illustrate the function of VISR filter rules, consider the input filter of the following example. The filter rule filters all incoming service requests for the occurrence of a parameter with the name `Member` and renames the parameter to Crew`member`.

```
<Filter type=”Input”>
  ...
  <Expression>
   <xsl:for-each select="Method/Parameters/Parameter/">
      <xsl:if test="@type=’Input’">
        <Crewmember><xsl:value-of select="Member"/></Crewmember>
      </xsl:if>
  </xsl:for-each>
  </Expression>
  ...
</Filter>
```

Mapping class. The `Mapping` class models declarative descriptions of view-registry mappings. These mappings are used to transform registry information into view based Web service descriptions, respectively into VISR service profiles. Mappings are implemented with XPath expressions. The mapping specification defines declarative mappings of different elements to the underlying registry data model. Table 3 summarizes the fields of the mapping class.

Property	Description
Element	Name of the view element that is mapped
Type	Type of mapping (generic or instance)
Link	Optional field containing link to external (XML) document
Expression	XPath Expression

Table 3. Available fields in the Mapping class

The view-registry mapping consists of two parts: (a) a generic mapping, based on XPath expressions, that specifies an abstract description and (b) a concrete instance based mapping of registry entries to VISR Web service profiles. The generic mapping can be considered as mapping that specifies classes of entries. The instance based mapping augments registry information with additional Web service Profile information that is not available in the registry. In the example below, we use all available types of mappings. The first portion of the example defines a generic mapping of the attribute `Name`. The second example shows the usage of default values. The third mapping example illustrates an instance mapping that augments a single instance of VISR service profile with information provided by an external resource. External resources are denoted by the `Link` element. This element identifies external accessible resources and acts as entry point for external resources.

```
<Mapping type=”generic”>
  <Element>
    <Name>Parameter</Name>
    <!—- select parameter name of service profile ->
    <Expression>
      //Parameters/Parameter/Name
    </Expression>
  </Element>
</Mapping>
...
<Mapping type=”default”>
  <!—- provide default value for element name ->
  <Element>
    <Name>Description</Name>
    <Expression>
      <Value>This is a service description</Value>
    </Expression>
  </Element>
</Mapping>
...
<!—- augment service description with existing text ->
<Mapping type=”instance”>
  <Element>Description</Element>
  <ID>infosys.tuwien.ac.at:8000:VSP:1</ID>
  <Link>http://www.infosys.tuwien.ac.at/treiber/service.xml</Link>
  <Expression>
    <!—- select Description of legacy service description ->
    //Service[@description='new']
  </Expression>
</Mapping>
```

Extension class. The `Extension` class acts as container for name-value pairs. These pairs model additional information that is not specified by the categories of VISR View profiles. This additional information is stored as name-value pairs. These pairs are assigned to the original registry entry and stored into the local repository. Extension instances provide a dynamic way to add arbitrary attributes to provider or Service instances. For example, if a Web service provider wants to add a “copyright” attribute to its profile, it can do so by adding an Extension with name “copyright” and value containing the copyright statement. The example below shows the use of the extension mechanism for the example above. Table 4 summarizes the fields if the extension class.

```
<Extension>
  <Element>
    <Name>Copyright</Name>
    <Value>2005 by Peter Pan Inc.</Value>
  </Element>
<Extension>
```

Property	Description
Name	Name of the element
Value	Assigned element value

Table 4. Available fields of the Extension class

2.3 VISR Service Profiles

Although related, VISR service profiles and VISR view profiles represent different problems and require different abstractions. Generally speaking, VISR service profiles provide a base for VISR views. VISR views depend of a common Web service description for the provision of Web service communities. Therefore, the main task of VISR service profiles is the integration of information from different Web service registries to a common lightweight Web service description. Figure 6 shows an example of the integration of UDDI registry entries and WSDL descriptions into VISR service profiles. Section 8 provides 2 examples for the integration of UDDI/WDSL and ebXML/WDSL Web service descriptions into VISR service profiles. Furthermore, the design of VISR service profiles is tailored to lightweight VISR peers, which offer limitations concerning memory and processing power (see Section 4).

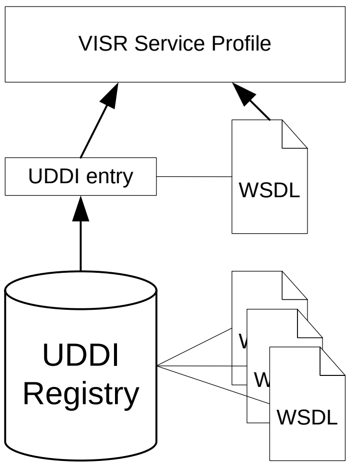

Fig. 6. VISR service profile overview

VISR service profiles consist of 5 categories:

Provider class. The `Provider` class contains information about the provider of the Web service. The provider class identifies the type of VISR peer and stores additional data about the Web service provider like name, address, description and URL. Table 5 summarizes the available fields of the provider class.

Property	Description
Name	The name of the Web service provider.
Phone	The telephone number of the Web service provider.
E-Mail	The email address of the Web service provider.
Fax	The fax number of the Web service provider.
Address	The postal address of the Web service provider.
URL	A URL where information about the Web service provider can be found.
Description	A brief, human readable description of the Web service provider.

Table 5. VISR provider profile

Service class. The Service class contains information about the Web services a provider provides. The Service profile contains the name of the Service, the invocation URL, a link to an external description files and the included methods with their parameters and corresponding descriptions (see Table 6).

Property	Description
Name	The name of the Web service.
URL	A URL where the Web service can be invoked.
Description	A brief, human readable description of the Web service summarizing what the Web service offers.

Table 6. General Web service data provided by VISR service profiles

Method class. The `Method` class encapsulates information regarding the methods a Web service provides. Method descriptions provide abstract descriptions of methods a Web service requestor invokes. These descriptions do not contain typing information for actual parameters, but provide the names of input and output parameters. Typing information is hidden from Web service requestors. Web service requestors only need to consider the abstract service descriptions to invoke the service. VISR Table 7 summarizes the available fields of the method class.

Property	Description
Name	The name of the method to invoke.
URL	A URL where the method can be invoked.
Description	A brief, human readable description of the method, summarizing what the method does.
Input	The input of the method specified as parameter name list
Output	The output of the method specified as parameter name list

Table 7. Abstract method description of Web service methods

Usage class. The `Usage` class is closely related to the method class. It provides how-to-use information about the service. This information includes an informal description and a formal description of the actual use of the Web service. The formal description provides information about the actual Web service invocation, the input and output parameters and an example of the usage. Consider our working example that illustrates the use of the usage class. In this example the usage is a formal description presented as request/result pairs of Web service invocations. In this example the usage is a formal description presented as request/result pairs of Web service invocations. The example shows the intended use of the hire method. The method hire has three input parameters, the name of the movie crew, and two dates. The abstract definition of the method is a follows:

```
hire(name, from, until)
```

A concrete example is the hiring of a movie crew with the name 'Peter':

```
hire(Peter, 26.04.04, 27.04.04)
```

The example below shows a concrete usage description of the hire method.

```
<Usage>
  <Invocation>
    <Description>
      The hiring of movie crews needs three parameters, the name of
      the Movie crew and two dates. The result is OK if the movie
      crew is hired or NOK if the movie crew is not hired.
    </Description>
    <Method>
      <Name>hire</Name>
      <Parameters>
        <Parameter type=”input”>
          <Name>from</Name>
          <Value>01.01.2004</Value>
        </Parameter>
        <Parameter type=”input”>
          <Name>until</Name>
          <Value>31.12.2004</Value>
        </Parameter>
       </Parameters>
    </Method>
  </Invocation>
  <Result>
    <Method>
      <Name>hire</Name>
      <Parameters>
        <Parameter type=”output”>
          <Name>result</Name>
          <Value>OK</Value>
        </Parameter>
      </Parameters>
    </Method>
  </Result>
</Usage>
```

3 VISR Architecture

The VISR Architecture is based on a peer to peer network that provides the infrastructure for the distributed nodes of the VISR network. Each node in the VISR peer to peer network plays a particular role. VISR distinguishes between three types of peers, (1) small scale or lightweight peers, (2) standard peers and (3) registry peers.

Lightweight peer. The Lightweight peer is usually a peer with limited memory and storage capacity, such as PDAs or Sub-notebooks. Lightweight peers are the typical transient members of the VISR network. A Lightweight peer joins and leaves the network at arbitrary points in time and provides usually a small number of Web services that are published at well known local addresses and are available only when the peer is part of the network. This type of peer can be considered as an anonymous peer in the network.

Standard peer. The Standard peer offers more memory and processing power than a lightweight peer. Usually a standard peer is a laptop or a typical desktop PC. This type of peer provides additional services, like notification and view services and hosts more Web services than a lightweight peer. The standard peer allows the user to name the peer. The name/URL combination of the peer serves as logical identification in the VISR peer network. The identification is needed when a view is published, because every stable public view acts as community and the publisher view maintains a list of all view members. In addition every member stores the name of the view originator and the identification of the view.

Registry peer. A registry peer is usually a gateway to a business registry with many Web service registry entries based for example on UDDI or ebXML. Registry peers transform requests of VISR peers to the registry and perform Web service registry data transformations into VISR service profiles.

Table 8 summarizes the functionality of the different peer types.

	Lightweight Peer	**Standard Peer**	**Registry Peer**
Web service Publishing	Yes	Yes	Yes
Web service Discovery	Yes	Yes	Yes
Membership	Transient	Transient, Stable	Stable
View Publishing	No	Yes	Yes
Peer Naming	No	Yes	Yes
Number of Services	1 – 5	5 – 25	Many

Table 8. VISR peer type features

3.1 VISR peer architecture

The VISR architecture is organized into three layers (see Figure 7). The registry layer integrates existing Web service registries using VISR service profiles. The view layer provides the means for the administration of views and the service layer provides the functionality for Web service publishing, Web service discovery and Web service provisioning.

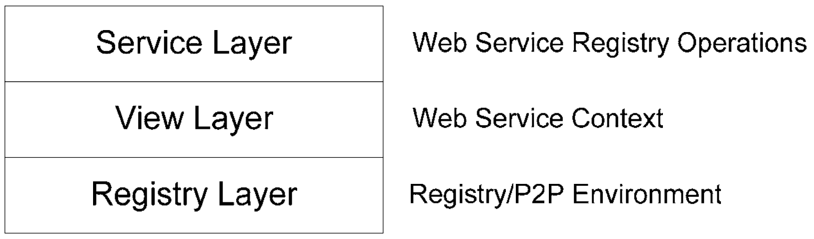

Fig. 7. VISR peer architecture overview

3.1.1 Registry Layer

The registry layer coordinates the integration of the different registries providing adapters for registry implementations and transient or transient Web service providers. The registry layer provides access to the underlying registry and acts as the lowest abstraction layer of the registry. It provides interfaces for the standard features (Web service discovery and Web service publishing) of the Web service registry. The registry layer distinguishes between two types of registry entries:

- **Stable registry entries.** A stable registry entry is a long-living registry entry that is created or deleted explicitly by a VISR peer or by the underlying registry. A VISR peer that publishes a stable Web service is also a stable peer that is a long term member of the network. The registry entry (i.e., the VISR service profile) is replicated among other stable peers to guarantee that the Web service description is available even when the Web service provider goes offline. If the VISR service profile augments a legacy registry entry then this data is replicated as well and stored into the local repository. Note that lightweight peers are not capable of publishing stable VISR registry entries, because lightweight peers are considered as transient members of the network. The other two types of peers, the standard and the registry peer are capable of creating stable registry entries.

- **Transient registry entries.** A transient registry entry is entirely specified by the VISR peer. A lightweight peer provides per default transient registry entries, the other two peer types may also provide transient registry entries. A transient registry entry is published implicitly as soon as the peer joins the network. If a peer leaves the network, the registry entry is not available any more. Note that all VISR core services are specified as transient Web services with corresponding VISR service profiles. The only difference to common transient registry entries is that a core VISR service cannot be deleted.

The registry layer (see Figure 8) maintains a repository that stores meta data about the registry and contains information about registry mappings that define the actual transformations of registry entries. Note, that Standard and Lightweight peers provide a local only provide a local repository with VISR Service Profiles without actual mapping information. Mapping information is needed by Registry peers for the transformation of existing registries. In this case the VISR Coordinator retrieves repository information from the Registry Wrapper directly and returns the result to the requestor.

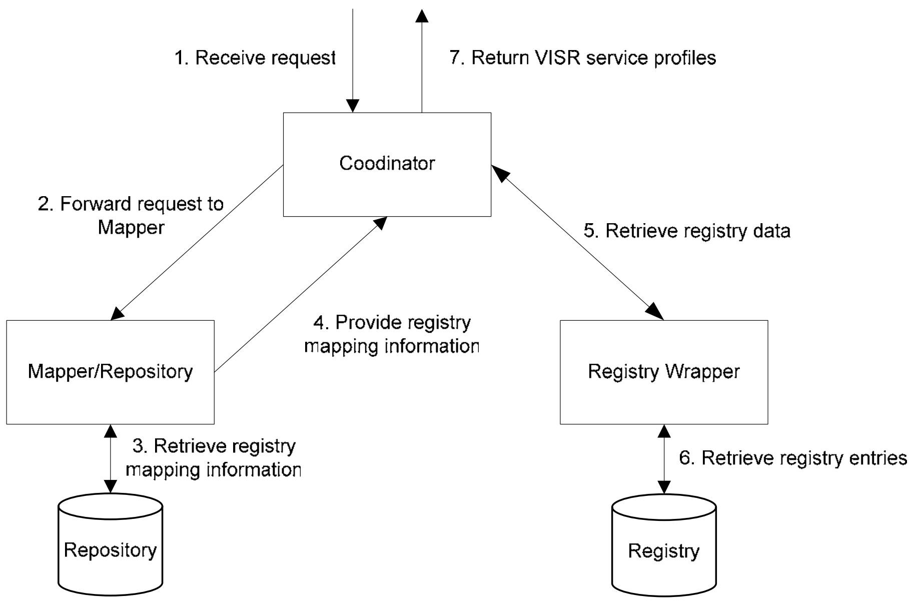
1. Receive request
7. Return VISR service profiles
Coodinator
2. Forward request to Mapper
5. Retrieve registry data
4. Provide registry mapping information
Mapper/Repository
Registry Wrapper
3. Retrieve registry mapping information
6. Retrieve registry entries
Repository
Registry

Fig. 8. VISR registry layer architecture of gateway peers.

Mapper/Repository. The role of Mapper/Repository depends on the type of VISR peer. If the peer acts as gateway to a Web service registry, the Mapper/Repository maintains information about the data format of the underlying Web service registry and the necessary transformations for the creation of VISR service profiles. Otherwise, the Mapper/Repository maintains a repository of VISR service profiles that are available on the corresponding VISR peer.

Registry Wrapper. The Registry Wrapper provides a standard interface for actual Web service registry implementations. It hides the “legacy” registry API behind the VISR registry API. The registry wrapper receives a request and maps the request onto a corresponding registry API call. For example, a registry wrapper may generate SOAP messages in order to query a public UDDI based Web service registry as depicted in Figure 9.

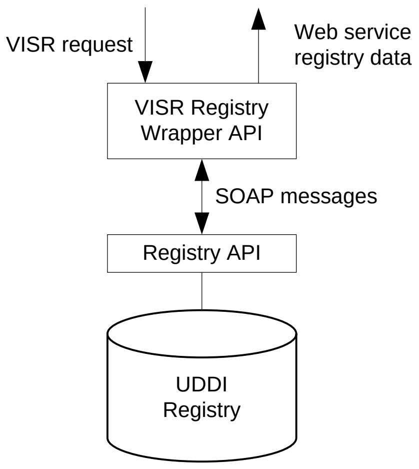

Fig. 9. VISR Web service registry wrapper. The VISR Web service API wrapper transforms requests (search service, etc.) into SOAP messages to access a public UDDI Web service registry.

Coordinator. The coordinator provides the actual Web service registry interface of VISR peers. It offers basic registry operations (publish, discover, etc.) and controls the actual Web service registry access. Depending on the type of VISR peer (gateway, standard, lightweight) the coordinator has two functions:

- **Coordination of legacy Web service registry access.** In order to access the "legacy" Web service registries and to the coordinator obtains information of the Mapper/Repository-component to access corresponding Web service registries. This information contains transformation/mapping instructions such as XSLT transformations or XPath expressions for the selection of Web registry entries.

- **Generation of VISR service profiles.** VISR service profiles are either generated by transformations of existing Web service registry content or accessing the VISR service profile repository.

3.1.2 View Layer

The VISR View layer provides the means necessary for the view concept. The View layer provides access to a repository that stores VISR view descriptions. The View layer has two related tasks. It transforms registry content that is provided as

VISR service profile according to view definitions and provides view based Web service access. The view based service access transforms the result of the service invocation into the view defined format (see Figure 10).

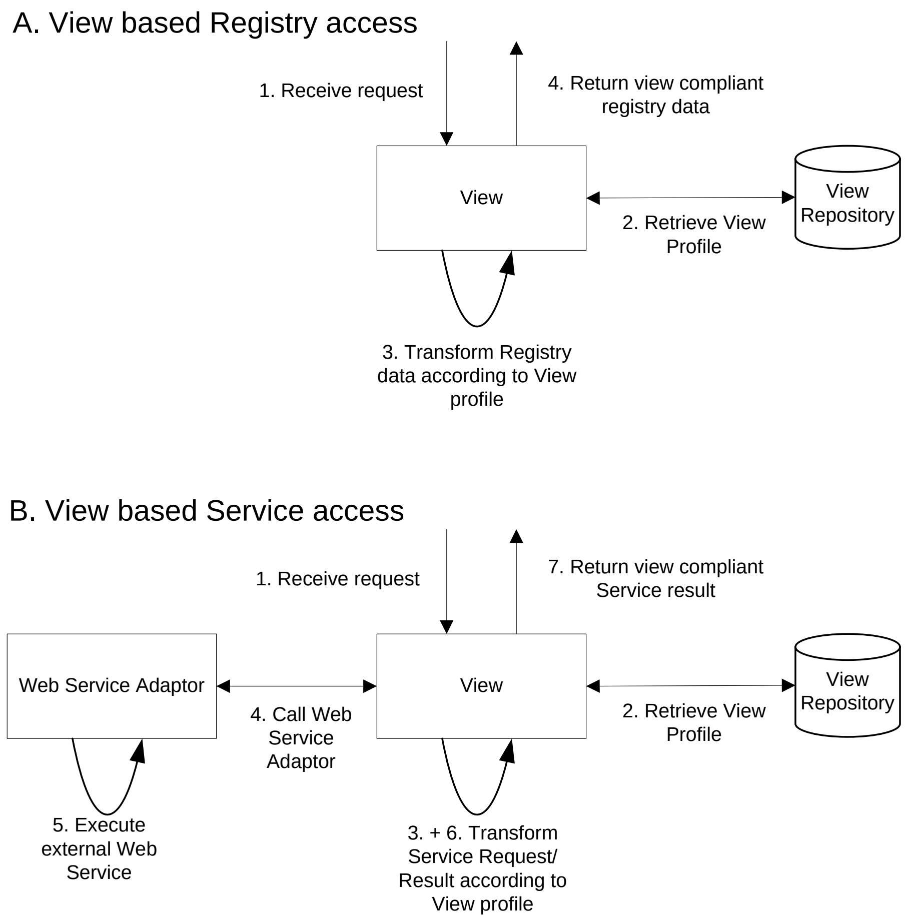
A. View based Registry access
1. Receive request
4. Return view compliant registry data
View
2. Retrieve View Profile
View Repository
3. Transform Registry data according to View profile
B. View based Service access
1. Receive request
7. Return view compliant Service result
Web Service Adaptor
4. Call Web Service Adaptor
View
2. Retrieve View Profile
View Repository
5. Execute external Web Service
3. + 6. Transform Service Request/ Result according to View profile

Fig. 10. VISR View Layer.

View based Web service discovery. The view based Web service discovery process operates in a view defined environment. A VISR view limits the search area of to view compliant Web services. Depending on the actual view specification, a view compliant Web service provides VISR service profiles, additional Web service information like for example quality of service attributes. Thus, a VISR view unifies two aspects of a Web service description. It provides both, high level Web service information like Web service provider name, address, etc., and concrete information about a Web service, like for example invocation URL or methods that can be invoked by Web service requestors.

Consider for example movie crews that possess (private) registries containing their service descriptions. A movie director that needs an overview about available services defines a view. Every movie crew is a member of the view is capable of serving search requests simultaneously and provides the director with corresponding VISR service profiles. Thus, using the view, a movie director does not to be aware of different registries.

View based Web service invocation. The view based Web service invocation under a view context leads to a unified and transparent Web service access. VISR Views define interfaces of the provisioned Web services. VISR Views use an abstraction that is similar to the WSIF [8] approach. WSIF introduces an abstraction layer over existing Web services thus providing Web service access that is not binding-dependent. WSIF uses an invocation strategy that allows invoking Web services directly using information provided by WSDL files. VISR also provides direct binding-independent access to Web services. VISR uses service profiles that provide definitions of the input and output parameters on an abstract level. 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.

To illustrate this approach, consider the working example with a view that defines a description for a salary interface. The view defines a membership for all movie crew members that are able to provide such a service, respectively are able to build a view conform wrapper for a salary service. A movie director who wants to control the costs of a stuntmen crew needs to invoke the salary service of every crew member and summarize the results. He does not need to take care of different Web service implementations or invocation protocols, because every VISR peer that provides a corresponding service performs the needed transformations. Figure 11 shows the interaction between the service requestor (movie director) and the service provider (stuntman).

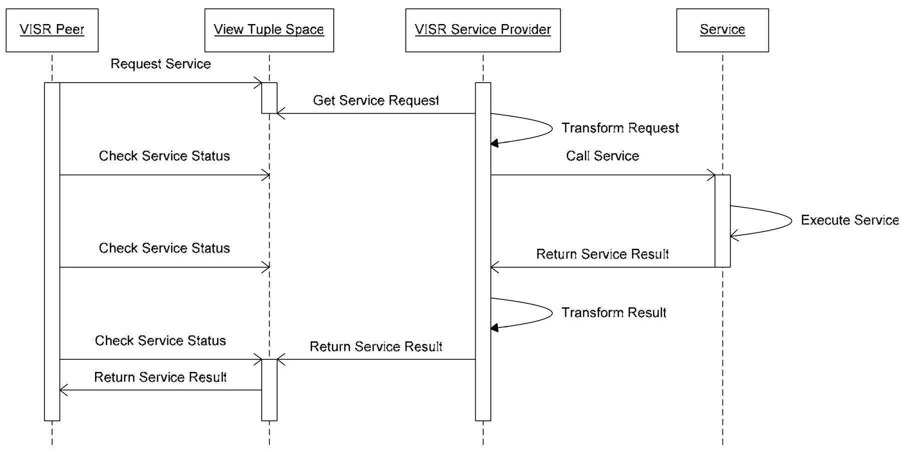

Fig. 11. View based service execution. The service requestor places a tuple in the view defined tuple space. Afterwards, the service requestor checks the tuple space periodically for the service result. Meanwhile, the service provider transforms the service request according to the view specification and executes the service. After service completion, the service provider places a tuple containing the result of the service invocation into the view defined tuple space that is finally retrieved by the service requestor.

3.1.3 Service Layer

The VISR Service Layer provides the core functionality for the basic operations of Web service registries. These services include a Web service Publishing Service, a Web service Discovery Service and a VISR view information service. These core services are available at every type of VISR peer. These services provide the basic functionality for the peer to peer registry and their descriptions are stored in the local repository of every peer. Table 9 summarizes all basic functions of the VISR peers.

Function	**Description**
createView(view)	Creates a view and publishes the view description. The description is stored in the local repository. This is function is not available on lightweight peers.
getViewDescription (id)	Returns the view description that is associated with the view identifier.
getUserProfile(id)	Returns the user profile of the VISR peer that is associated with the peer identifier.
getProviderProfile(id)	Returns the provider profile of the VISR peer that is associated with the peer identifier.
joinView(id)	Performs the login procedure of a view.
leaveView(id)	Performs the logout procedure of a view.
search (criteria)	Searches globally for the given criteria based on the VISR Service Profile and returns a list of adequate Web services.
publishService(id, service)	Publishes a service in a view.
unpublishService(id, service)	Removes a service from a view.
searchView(id, type, criteria)	Searches for the given criteria within a view and returns a list of adequate Web services.
matchService(service, view)	Checks if a service description is compatible with the view.
getMethodProfiles(id)	Returns all method profiles of a given Web service.
getServiceUsage(id)	Returns the usage description of a Web service.

Table 9. VISR core functions

4 Related Work

The ebXML 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, whereas our work provides a more flexible approach, because it allows the creation of long lasting stable views and also transient views. Transient 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 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.

The METEOR-S [15] project uses an upper ontology for the integration of distributed registries. VISR also provides the means of integrating Web service registries into a common Web service registry, but does not provide an ontology based approach. VISR uses common Web service descriptions (VISR service profiles) to create uniform accessible Web service registries. Furthermore, METEOR-S proposes a lightweight Web service description language (WSDL-S) that extends WSDL 2.0 service descriptions with semantic annotations. VISR is also capable of extending current Web service descriptions with additional data, but in comparison with METEOR-S, VISR does not focus on semantic issues. VISR omits semantics concerning Web service description in favor of implementation issues concerning lightweight peers with limited processing power and memory capabilities.

The work presented in [11] uses DAML-S [17] profiles to represent semantic meta data of UDDI registry entries. This approach uses UDDI registries with tModels to link to DAML-S profiles that are stored separately. VISR does not store information in the existing registry. It leaves the original information unchanged and provides gateways to distributed registries. Like in [11] additional data is stored separately from the original Web service registry entries in repositories that are no part of the original Web service registry. The semantic meta data is used for Web service retrieval but does not cover the issue of different web service registry implementations. [11] focuses on UDDI and implements a centralized approach and does not consider distributed registries like VISR.

In contrast to OWL-S [], which superseded DAML-S, VISR service profiles take a "pragmatic" approach without following workflow issues –like, for example, Web service orchestration or Web service composition. VISR service profiles offer a lightweight Web service description that focuses on interface descriptions of Web services.

The Webtransact framework [12] presents an infrastructure for the integration of heterogeneous Web services. Our work encompasses contributions from this work, since VISR provides unified Web service invocation, similar to the concept presented in [12]. Webtransact uses Web service mediators to invoke different Web services. In VISR, the notion of the registry peer is similar to the Web service mediator of the Webtransact infrastructure. In contrast to VISR, the Webtransact framework does not consider distributed registries or different Web service registry implementations.

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. WSDA proposes WSIL [20] containers for the actual service description. VISR borrows the idea of tuple spaces from WSDA. 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 [9, 16] 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 transient provider selection as SELF-SERV does.

VISR proposes a community concept similar to the WebBis [21] communities. WebBis offers two types of communities, push and pull communities. These communities correspond roughly to transient respective stable 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.

Pilioura et al [27] propose an integration concept of distributed Web service registries based on semantic meta data. In comparison with VISR, the PYRAMID-S architecture operates on a higher abstraction level, using gateways as entry points for Web service registries. In contrast to our work, PYRAMID-S does not consider transient Web service providers. VISR provides explicit support for transient Web service provider, by the provision of decoupled communication means based on tuple spaces.

5 Conclusion and further work

We present concepts for the integration of heterogeneous Web service registries and transient Web service providers. In our approach we consider a peer to peer network as a feasible solution for the integration of different Web service registries. We distinguish between three types of peers, depending on their availability and processing power. Lightweight peers are considered as transient members of the VISR peer to peer network which join and leave the network dynamically. They are devices that offer limited processing power and memory capacity. This type of peer provides a small number of Web services and stores Web service descriptions locally. Standard peers offer more Web services and more processing capabilities. Standard peers provide the means for the creation of abstract contexts for Web service registry entries (views). They are considered as stable members of the network with high availability. Like lightweight peers, Standard peers also store registry information locally. Registry peers act as gateways to different registry implementations and provide a mediation service for the invocation of Web services through the VISR peer to peer network. They are considered as building blocks of the VISR network and provide high availability and processing power.

A significant part of this paper discusses the view concept of VISR. Views are abstract contexts in which Web services are published and can be regarded as virtual registries. Views allow the creation of Web service communities that provide related Web services with similar or equal service invocation. In addition, Views provide the means for invocating Web services in a unified way using the VISR communication system. In out approach, views include information that is out of the scope of common Web service registry entries. A view consists of abstract input and output descriptions and internal mappings to registry information. The underlying registry data model remains, all references to additional information are stored outside the original registry. Views involve VISR service profiles that act as wrapper for existing registry entries. VISR service profiles expand available registry information with abstract descriptions of Web service operations. The operational information consists of two parts, abstract method specifications and actual usage descriptions. Usage descriptions serve as real world examples for the use of Web services. These descriptions include method sequences with concrete parameters and examples of corresponding results. In this context, we generally treat Web services as black boxes that provide input and output data. Our focus lies on the extraction of relevant information from Web service descriptions to provide a structural description of the operations a Web service provides. Our work transforms WDSL descriptions into the operational part of VISR service profiles. This abstract description is used for our simple matching algorithm that uses structural information for the matching of relevant Web service descriptions. Another aspect of views is the seamless integration of differing registry data models. The integration provides a unified and simultaneous access of distributed registries. We use declarative descriptions to create mappings from views respectively VISR service profiles to existing registry entries. If declarative mappings are not possible we use a programmatic solution based on a plug-in model to create the appropriate transformations. So far, this technique does not involve the use of semantic information, because this is considered to complicate the design and implementation of lightweight peers.

A prototype implementation of VISR in Java is under way. We are using IBM's tuple space implementation TSpaces as primary communication and storage means. The tuple space concept provides the means for decoupled operation in time and space.

The implementation of a matching algorithm of VISR service profiles is also in development. We are considering an algorithm that uses structural matching for the candidate selection of VISR service profiles. After the completion and the performance analysis of the prototype we are going to extend the functionality of the prototype. We consider (semi-) automated provider and Web service selection based on structural similarity of VISR service profiles. Since the current tuple space implementation is centralized, we intend to use a distributed version tuple spaces for our implementation of the registry layer to improve the flexibility of our approach.

6 References

[1] Universal Description, Discovery and Integration: UDDI Technical White paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf. 2000

[2] OASIS/ebXML registry services Specification v2.5. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf. 2003

[3] OASIS/ebXML Technical Architecture Specification. http://www.ebxml.org/specs/ebTA.pdf. 2001

[4] OASIS/ebXML registry Information Model v2.5 http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrim-2.5.pdf. 2003

[5] Business Process Specification Schema. http://www.ebxml.org/specs/ebBPSS.pdf. 2001

[6] ebXML Collaboration-Protocol Profile and Agreement Specification. http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf. 2002

[7] Keith Ballinger, Peter Brittenham, Ashok Malhotra, William A. Nagy, Stefan Pharies. Web services Inspection Language (WS-Inspection) 1.0. IBM, Microsoft. 2001

[8] Matthew J. Duftler, Nirmal K. Mukhi, Aleksander Slominski and Sanjiva Weerawarana. Web services Invocation Framework (WSIF). IBM T.J. Watson Research Center. 2001

[9] Boualem Benatallah, Marlon Dumas, Quan Z. Sheng, Anne H.H. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web services. The 18th IEEE International Conference on Data Engineering (ICDE02), pp 297-308, San Jose, California, USA, Feb, 2002.

[10] Anbazhagan Mani, Arun Nagarajan. Understanding quality of service for Web services. http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html. 2002

[11] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne1, Katia Sycara. Importing the Semantic Web in UDDI. LNCS 2512, pp. 225–236, 2002. Springer-Verlag Berlin Heidelberg 2002

[12] Paulo F. Pires, Mário R. F. Benevides,Marta Mattoso. Mediating Heterogeneous Web Services. Computer Science Departament, Núcleo de Computação Eletrônica – NCE2, System Engineering and Computer Science Program – COPPE3. Federal University

of Rio de Janeiro. 2003
[13] Joseph M. Chiusano, Booz Allen Hamilton, Farrukh Najmi, Sun Microsystems. Registering Web services in an ebXML Registry, Version 1.0. 2003
[14] John Colgrave, Karsten Januszewski. Using WSDL in a UDDI Registry, Version 2.0. http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm. 2003
[15] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, John Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web services. Information Technology and Management 6, 17–39, 2005. Springer Science + Business Media, Inc.
[16] Quan Z. Sheng, Boualem Benatallah, Yan Q. Zhu, Rayan Stephan, Eileen Oi-Yan Mak. Discovering E-services Using UDDI in SELF-SERV. Proceedings of International Conference on e-Business (ICEB2002), Beijing Institute of Technology Press, Beijing, China, 2002, pp. 396 - 401
[17] DAML-S Coalition. DAML-S: Web services Description for the Semantic Web. ISWC 2002, LNCS 2342, pp. 348–363, 2002. Springer-Verlag Berlin Heidelberg 2002
[18] Wolfgang Hoschek. Peer-to-Peer Grid Databases for Web service Discovery. CERN IT Division. 2002
[19] UDDI Spec Technical Committee Specification. UDDI Version 3.0.1. 2003
[20] Keith Ballinger, Peter Brittenham, Ashok Malhotra, William A. Nagy, Stefan Pharies. Web services Inspection Language (WS-Inspection) 1.0. IBM, Microsoft. 2001
[21] Brahim Medjahed. Boualem Benatallah. Athman Bouguettaya. Ahmed Blmagarmid 2004).WebBIS: An Infrastructure for agile Integration of Web Services. International Journal of Cooperative Information Systems, Vol. 13, No. 2 (June 2004), pp. 121-158.
[22] Schahram Dustdar, Martin Treiber. A View Based Analysis on Web service Registries. Distributed and Parallel Databases, forthcoming
[23] W3C. XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999. http://w3c.org/TR/xpath. 1999
[24] Aphrodite Tsalgatidou and Thomi Pilioura. An Overview of Standards and Related Technology in Web Services. Distributed and Parallel Databases, 12, 135–162, 2002. Kluwer Academic Publishers. 2002
[25] W3C. XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116. 1999
[26] IBM. TSpaces. http://www.alphaworks.ibm.com/tech/tspaces. 2003
[27] Thomi Pilioura, Georgios-Dimitrios Kapos, and Aphrodite Tsalgatidou. Seamless Federation of Heterogeneous Service Registries. LNCS 3182, pp. 86–95, 2004. Springer-Verlag Berlin Heidelberg 2004

7 Appendix

7.1 UML diagram of VISR View Profile

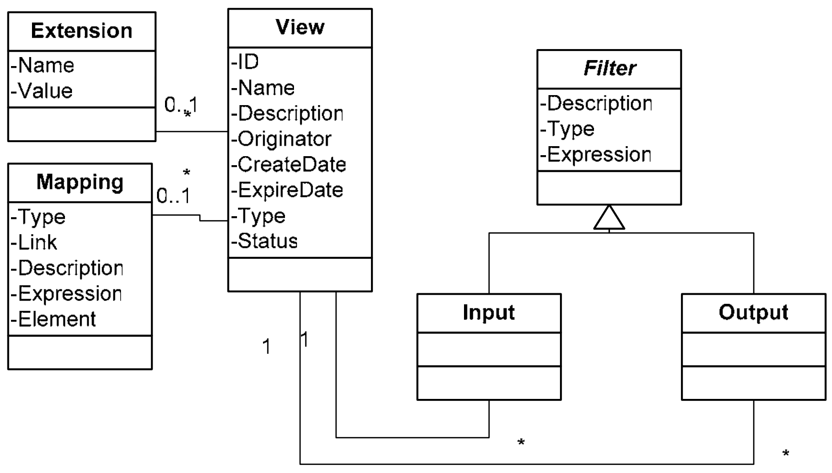

Fig. 12. UML diagram of VISR view profile

7.2 UML diagram of VISR Service Profile

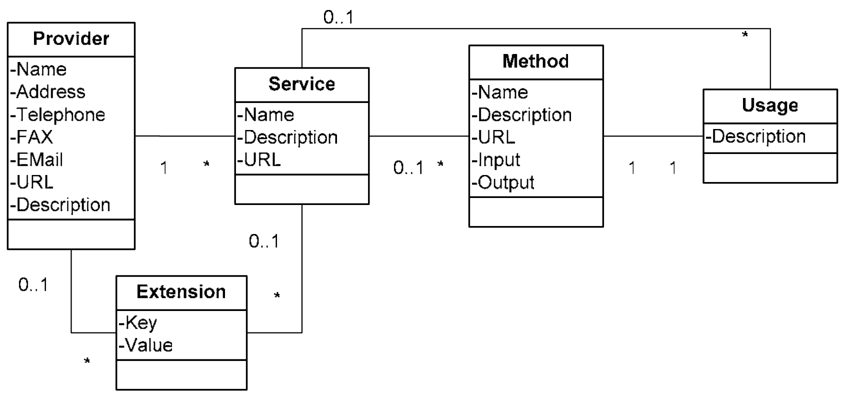

Fig. 13. UML diagram of VISR service profile

7.3 Registry mappings

This section contains examples for generic mappings of ebXML and UDDI registry entries of VISR service profiles. Our prototype implementation uses XPath [23] and XSL transformations to specify the mappings between the elements of VISR service profiles and UDDI/ebXML/WSDL elements.

7.3.1 UDDI

Figure 14 illustrates the generic mapping to UDDI entries and their corresponding WDSL files. Note, that the detailed description is retrieved with the tModel structure as proposed in [14]. The BusinessEntity structure provides the basic information needed by the VISR Web service provider and is mapped directly on the VISR Web service provider class. Web service information (name, URL, description) is available in the BusinessService structure and is mapped to the VISR Web service class. The link provided by the BusinessTemplate structure points to the external WSDL file. The operation, input and output descriptions of the WSDL file are mapped to the description of the methods in the VISR service profile.

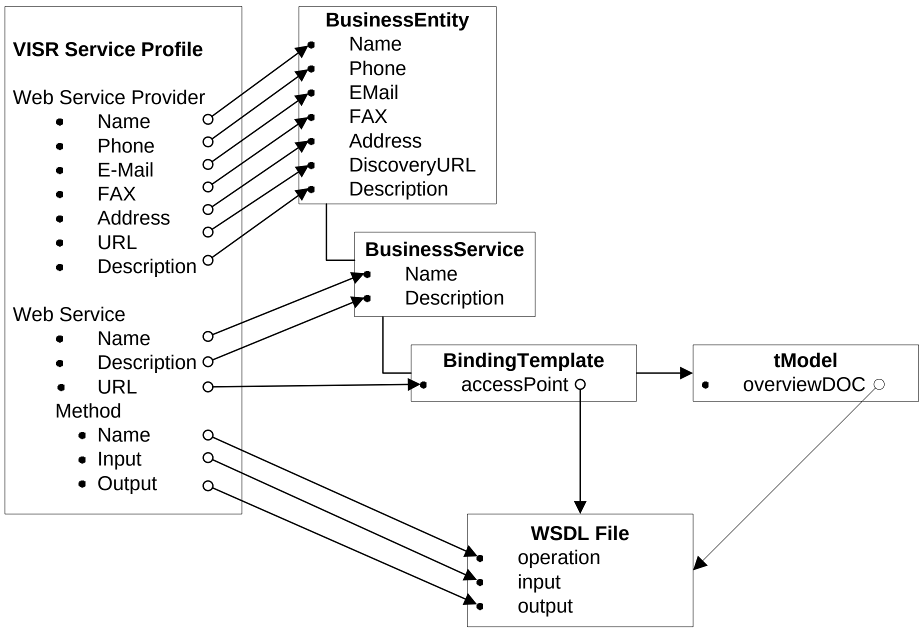

Fig. 14. VISR Service Profile to UDDI Mapping

VISR uses XSL transformations to generate VISR service profiles from UDDI information. The example below shows the transformation of the operational part of WSDL files to VISR interface description of VISR service profiles.

7.3.2 ebXML

The mapping of the VISR service profile and the ebXML data model is similar to the UDDI registry mapping. The difference lies in the different data model. The ebXML data model is different, Web service information is stored in a different format. We assume that a Web service is stored in the ebXML registry as specified in [13], where actual Web service description is stored as external WSDL file. Figure 15 illustrates the VISR Service Profile mapping of the ebXML registry model. The ebXML Organization class provides the same basic information as provided by the VISR Web service provider and is mapped directly on the VISR Web service provider class. Web service information (name, URL, description) is available in the Service class of the ebXML data model. The Service class also contains a link to the external WSDL file. The Service class is mapped to the VISR Web service class. The link provided by the attribute SpecificationLink points to the external WSDL file, its method descriptions are mapped as in the UDDI example, using the operation, input and output descriptions of the WSDL file for the description of the methods in the VISR service profile. Note that the associations between the registry objects are not explicitly shown.

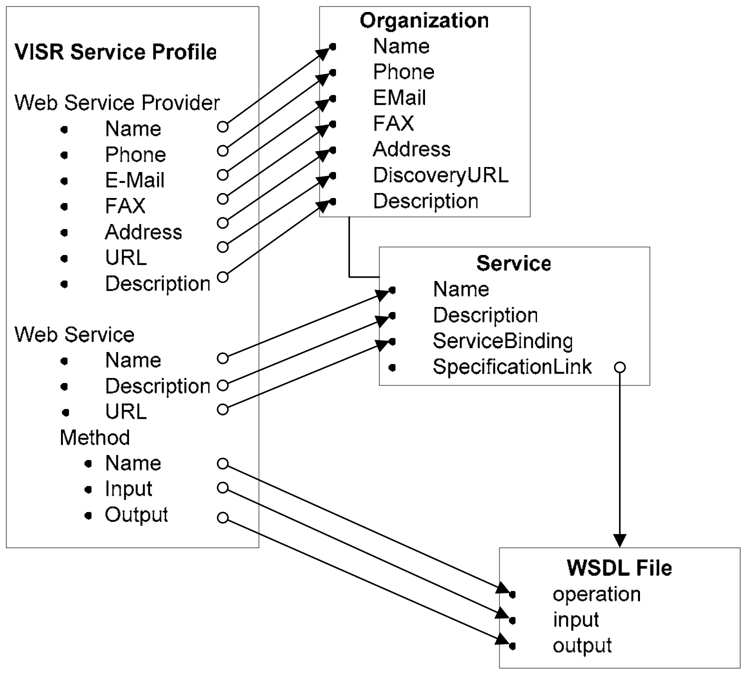

Fig. 15. VISR Service Profile to ebXML Mapping