Technical University of Vienna
Information Systems Institute
Distributed Systems Group

# WiZNet

*Integration of different Web service Registries*

Schahram Dustdar and Martin Treiber
dustdar@infosys.tuwien.ac.at
e9426464@student.tuwien.ac.at

TUV-1841-2004-18 September 8, 2004

*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 WiZNet 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. WiZNet 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

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

# WiZNet – Integration of different Web service Registries

Schahram Dustdar and Martin Treiber

Distributed Systems Group
Vienna University of Technology
{dustdar@infosys.tuwien.ac.at|E9426464@student.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 WiZNet 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. WiZNet 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 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.

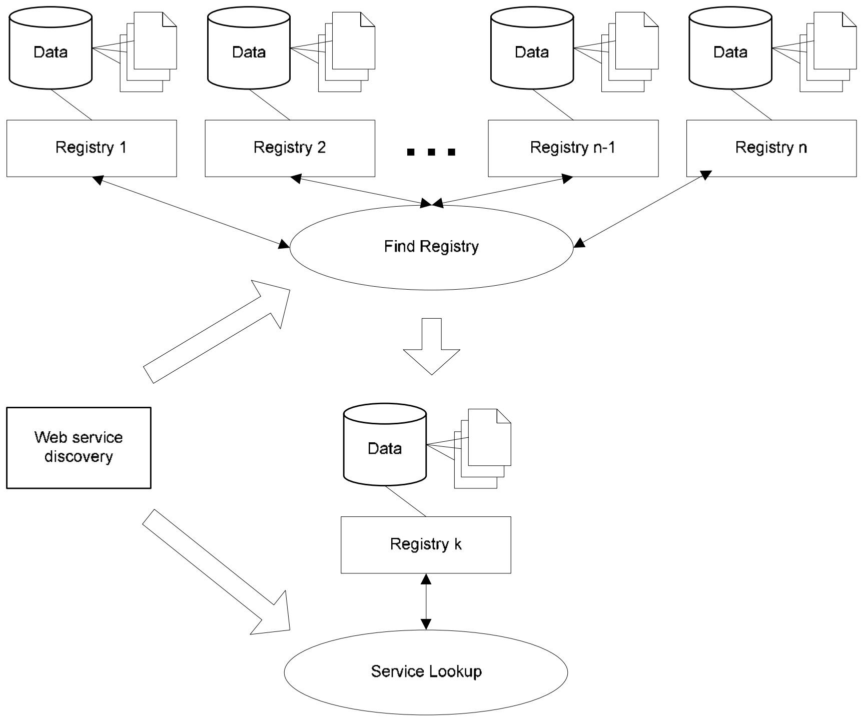

**Fig. 1.** Web service discovery in distributed registries

However, as Web services become more dynamic, consistency between distributed registries cannot be guaranteed any more. Consider a dynamic Web service `S` that registers itself in a registry `A`. If this registry is replicated before the dynamic 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. These registries usually provide related services from different providers. Consider for example stuntmen crews that offer their services in private registries that are specialized for stuntmen crews. Hence the search for stuntmen services involves searches in different Web service registries, before finding the adequate Web service (see figure 1).

To avoid these limitations, we propose WiZNet. The WiZNet architecture provides a peer to peer network of interconnected Web service registries and dynamic Web service providers (see figure 2). The WiZNet peer to peer architecture offers a meta data model that provides an integration concept that unifies these different Web service registries and dynamic 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 dynamic 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.

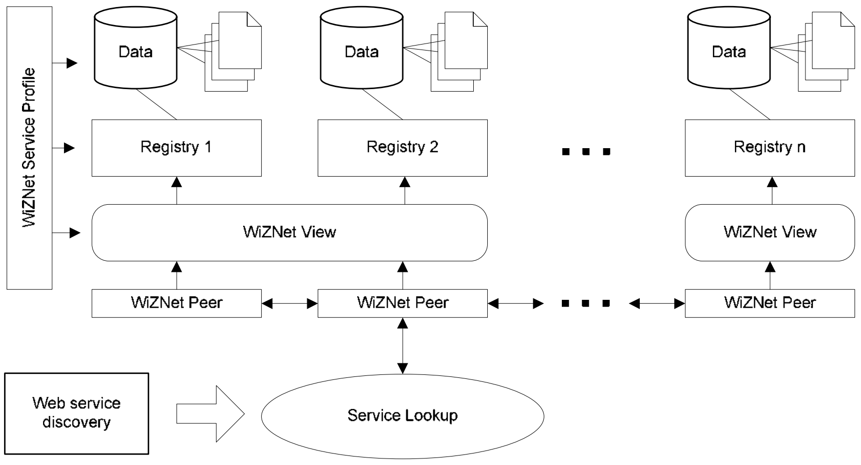

**Fig. 2.** WiZNet Overview

The remainder of the paper is organized as follows. Section 2 presents related work. Section 3 introduces a working example that is used throughout the paper. Section 4 illustrates the WiZNet peer to peer architecture. Section 5 presents the WiZNet View concept. Section 6 gives an example for the integration of UDDI and ebXML registries. Section 7 explains core WiZNet Services. Section 8 concludes the paper.

## 2 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. WiZNet 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 static views and also 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. WiZNet 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 WiZNet peers WiZNet 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, WiZNet offers a more flexible approach. WiZNet 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. In comparison with WiZNet, METEOR-S focuses on a higher level of integration whereas our work is on a lower level, i.e. its main focus is to unify the different registry data models. WiZNet focuses on the declarative integration of distributed registries with regard to their differing data models. WiZNet provides the means to create logical registries (views) that provide related registry entries. In contrast to METEOR-S that uses semantic meta data to create ontologies with related registry entries, WiZNet does not follow a semantic approach. Semantic issues are currently not part of our work, because this is considered as additional administrative overhead that would compromise our goal of integrating lightweight peers.

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. WiZNet 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 WiZNet.

The Webtransact framework [12] presents an infrastructure for the integration of heterogeneous Web services. Our work encompasses contributions from this work, since WiZNet provides unified Web service invocation, similar to the concept presented in [12]. Webtransact uses Web service mediators to invoke different Web services. In WiZNet, the notion of the registry peer is similar to the Web service mediator of the Webtransact infrastructure. In contrast to WiZNet, 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 WiZNet, 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. WiZNet borrows the idea of tuple spaces from WSDA. Tuples are used to represent WiZNet service profiles of transient clients. In comparison with WiZNet, WSDA does not consider Web service communities or different Web service registry implementations.

WiZNet 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. WiZNet follows a similar idea, regarding the structuring of communities, respectively views, but WiZNet does not provide dynamic provider selection as SELF-SERV does.

WiZNet 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 dynamic respective static views of WiZNet. In comparison, WiZNet 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. WiZNet also provides common Web service descriptions that are specified by WiZNet service profiles. WiZNet service profiles expose some similarities to WebBis service wrappers but operate on a different level. WiZNet 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.

## 3 Working Example

This section introduces the working example that is used in the rest of the paper. It illustrates a possible scenario for the use of WiZNet. In this scenario, a movie director has to coordinate several different film crews. Each of the film crews offers a set of services that are needed for the making of the movie. Some of the film crews work in a rather loosely coupled way with the film director. They provide their services only on demand for a certain time. Consider for example a stuntmen crew coordinating its activities on the film set with the help mobile devices. Every stuntman offers his expertise on certain topics as a service. The service description may be stored in a stuntmen registry that is administered by a stuntmen community or a company that provides this type of services. Other film crews may also provide private registries where they store their service descriptions. In order to coordinate these different teams the film director must search different registries to find all necessary services from the film crews. WiZNet offers unified access to these different Web service registries and also offers unified Web service invocation.

## 4 WiZNet Architecture

This section describes the WiZNet architecture in detail. The WiZNet Architecture follows a distributed approach. WiZNet is organized as a peer to peer network where each peer acts either as a gateway to a registry or as a Web service provider. Every peer offers the same basic functionality, regarded as WiZNet core functionality. The WiZNet core functionality provides interfaces for Web service publishing, Web service discovery, Web service provisioning, WiZNet Views, and WiZNet peer to peer communication. These separate interfaces are organized into a layered architecture as depicted in figure 3.

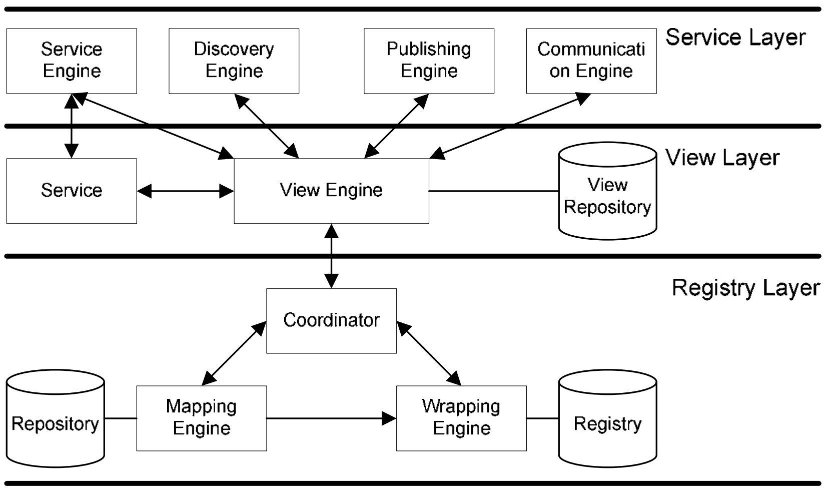

**Fig. 3.** WiZNet peer architecture overview

The registry layer coordinates the integration of the different registries providing adapters for registry implementations and meta data about registries. The Registry Layer consists of three separate modules, the WiZNet Mapping Engine, the WiZNet Wrapping Engine and the WiZNet Communicator Engine (see figure 3).

The WiZNet Wrapping Engine provides access to the underlying registry. This module is 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 and the means for the view based access to registry data. View based registry access filters the registry data according to a view definition and transforms registry information according to a view specification (see section 5).

The WiZNet Mapping Engine maintains a repository that stores meta data about the registry and contains information about view-registry mappings that define the actual transformations of registry entries. These mappings are organized as generic WiZNet service profiles and instances of WiZNet service profiles. Generic WiZNet service profiles define classes of Web services and are used for the transformation of registry entries into instances of WiZNet service profiles. Instances of WiZNet service profiles provide value added information of existing registry entries. Both types of mappings are discussed in detail in section 5, the WiZNet service profile is discussed in detail in section 7.

The WiZNet Coordinator coordinates activities between the Mapping Engine and the Wrapping Engine (see figure 4). Upon receiving a request it forwards the request to the Mapping Engine. The Mapping Engine parses the request and checks its repository for registry mapping information. The registry Mapping Engine provides mapping information for registry data retrieval to the Wrapping Engine. In addition, the Mapping Engine returns matching WiZNet service profiles to the Coordinator. The Wrapping Engine retrieves registry information and transforms the result according to the WiZNet service profiles. The result of this transformation is returned to the Coordinator. The Coordinator augments WiZNet service profiles with registry data and returns the data. Note, that lightweight clients provide no local registry. Lightweight peers only provide a local repository with WiZNet Service Profiles. In this case the WiZNet Coordinator Engine retrieves repository information from the Mapping Engine and returns the result to the requestor.

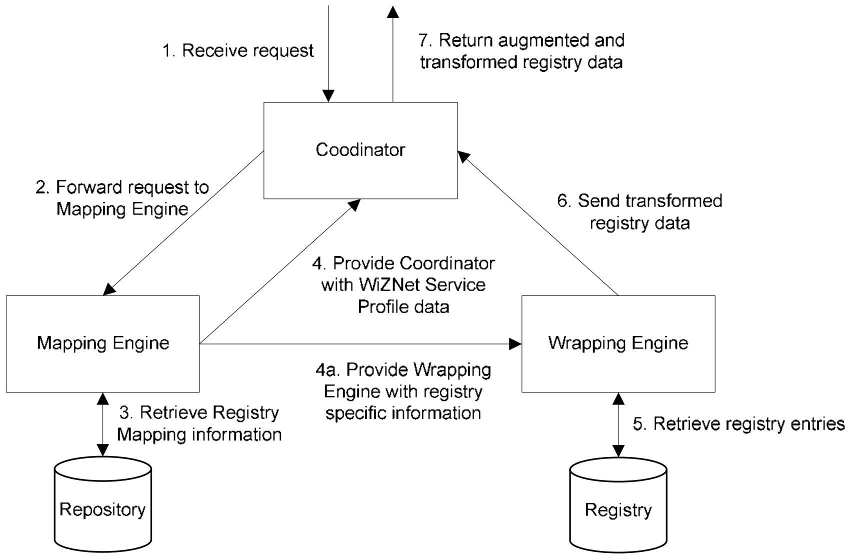
1. Receive request
7. Return augmented and transformed registry data
Coodinator
2. Forward request to Mapping Engine
6. Send transformed registry data
4. Provide Coordinator with WiZNet Service Profile data
Mapping Engine
Wrapping Engine
4a. Provide Wrapping Engine with registry specific information
3. Retrieve Registry Mapping information
5. Retrieve registry entries
Repository
Registry

**Fig. 4.** WiZNet registry layer

The WiZNet View layer provides the means necessary for the view concept. The View Layer consists of two modules, the View Engine, and the Service module (see figure 5). The View Engine provides access to a repository that stores view descriptions (see section 5). The View Engine has two related tasks. It transforms registry content that is provided as WiZNet service profiles according to view definitions and provides View based Web service access. Depending on the type of the view, these transformations can either be performed with the help of declarative descriptions or with plug-ins that perform the needed transformations. When using declarative transformation descriptions for the retrieval of Web service registry data, the View Engine provides WiZNet View profiles to the Coordinator. The Coordinator provides declarative view descriptions and returns adequate registry entries. If a plug-in is used, the Coordinator executes the plug-in and returns the result to the View Engine. In either case the View Engine returns the result to WiZNet Service Layer, respectively to one of the modules of the WiZNet Service Layer. The WiZNet View Layer also provides view based Web service access. In this case, the View Engine provides the Service with View data. The Service acts as wrapper for the Web service. It receives the requests and transforms the requests according view specifications and calls the Web service. The result of the Web service invocation is returned to the View Engine. The View Engine transforms the result of the Web service invocation returns the result to the requestor.

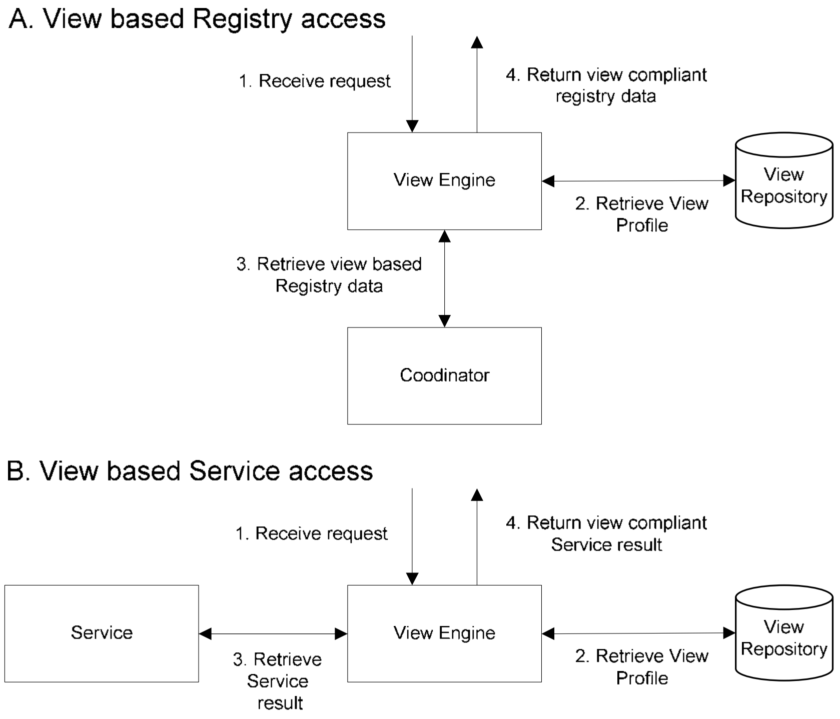
A. View based Registry access
1. Receive request
4. Return view compliant registry data
View Engine
View Repository
2. Retrieve View Profile
3. Retrieve view based Registry data
Coodinator
B. View based Service access
1. Receive request
4. Return view compliant Service result
Service
View Engine
View Repository
3. Retrieve Service result
2. Retrieve View Profile

**Fig. 5.** WiZNet View Layer

The WiZNet Service Layer provides the core functionality for the basic operations of Web service registries. It consists of four modules, the Service Engine, the Publishing Engine, the Discovery Engine and the Communication Engine. The

Service Engine provides the interfaces for Web services of a WiZNet peer. The Service Engine provides either, direct access to the Web service or view based access to the Web service. In the latter case, the Service Engine uses the View Engine for the transformation of the input and output of the Web service according to the corresponding view specifications. If the WiZNet peer is a lightweight client, the direct view based Web service access is not available, since lightweight peers are considered as transient WiZNet peers that do not provide the needed memory and processing power. In this case, another peer may perform the needed transformations acting as view proxy.

The WiZNet Publishing Engine offers an interface for the publishing of local Web services. The WiZNet Publishing Engine implements a Publish/Subscribe approach. Every WiZNet peer listens by default for the publishing of Web services, i.e., every WiZNet peer has a subscription for events that mark the publishing of a Web service. When a peer publishes a Web service in the context of a view (see section 5), the Publishing Engine sends a notification message that informs the view originator about the new Web service. The notification contains information about the nature of the Web service, i.e., whether it is a permanent service or the service is a temporarily available service and meta data regarding view information. If the Web service matches the view description the WiZNet peer becomes a member of the view.

The Communication Engine provides the means for the WiZNet communication protocol. The Communication Engine is responsible for the correct joining and leaving of the WiZNet peer network. The Communication Engine offers a notification service, whereby every WiZNet peer is able to register for arbitrary events, including for example the joining of new peers, etc. Another task is the administration of the neighbor table that stores a list of all connected neighbors. The neighbor table is needed for the forwarding of messages.

The WiZNet Discovery Engine manages retrieval of registry information. Like the Service Engine, the Discovery Engine varies according to the type of WiZNet peer. A lightweight peer provides a local Web service description based on the internal Web service registry profile on a well known port. Since lightweight peers provide no view information, queries containing view information are delegated to proxy peers that are able to perform the needed transformations. Standard WiZNet peers and WiZNet registry peers check search requests for view information first. The view information is used for the mapping of the registry entries to the view. If the query is executed successful, the result is returned to the requestor. If no adequate Web service can be found, a “service not found” message is sent back to the search requestor.

### 4.1 WiZNet peer types

WiZNet supports three types of peers, namely Lightweight peers, Standard peers and Registry peers. Every peer type offers the basic functionality but different additional features. 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 WiZNet 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. 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 WiZNet peer network. The identification is needed when a view is published, because every static 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. The third type of WiZNet peer is the 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. The registry peer acts as a Web service mediator for the view based Web service invocation (see section 5) because the actual Web service providers are no members of the WiZNet network and do not provide the functionality of WiZNet peers. The Registry peer transforms the requests of the WiZNet peers to the Web service provider when a Web service is executed. Like the Standard peer the registry peer provides wrappers for different registries respectively different APIs that are conform to the basic functions of the WiZNet peers. Table 1 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 |
| Web service Mediation | No | No | Yes |
| Notification Service | No | Yes | Yes |
| Membership | Dynamic | Dynamic, Static | Static |
| View Publishing | No | Yes | Yes |
| Peer Naming | No | Yes | Yes |
| Number of Services | 1 – 5 | 5 – 25 | Many |

**Table 1.** WiZNet peer type features

## 5 WiZNet View Concept

This section discusses the WiZNet view concept. A WiZNet view defines an abstract context that specifies how Web services are published, discovered and invoked by peers of the WiZNet peer to peer network. The context provides additional information about Web services that are out of the scope of single Web service descriptions. Consider the working example with view information about the movie project, defined by the director of the movie. A Web service provider that publishes view 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 view, 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 Web service description.

The example illustrates two different roles that are defined by the WiZNet view concept. The view originator (VO) specifies a view description and publishes the view description. The view implementer (VI) receives a view description and decides to implement the view specification. The implementation uses local mappings and transformations for the underlying registry implementation to transform the registry information accordingly. Note, that the view originator may be the view implementer at the same time.

A view may either exist independently without relation to other views or relate to other views with the help of basic set operations. These operations include union, subset and intersection. The result of a set operation is a new view that puts every included Web service into a new context. The context depends on the type of the set operation. Consider for example two views, called `movie mayhem` and `movie senseless`. Both views are created by a film director and provide information about two movie projects of the film director. We now assume that the film director needs a new view called `movie overall` that includes both movie projects with all services. The new view is then created with the union operator and provides both view descriptions and all Web service descriptions that are available in one of the two views.

## 5.1 WiZNet communities

The view concept allows the definition of Web service communities [9] that integrate similar Web service registry entries to a community. A WiZNet community is essentially a group of related services. These services provide an abstract description, i.e., the view description and may offer identical service interfaces. This allows an abstract specification of services without regarding to the actual Web service provider and the actual Web service. The Web service provider must implement the abstract method with mappings and filters. Consider the stuntmen community of our working example. For coordination purposes, every stuntman may offer a notification method when he finishes his work on the film set. In our example there are different types of stuntmen with different devices (PDA, Laptop) on the set. If every stuntman has his own device dependent notification method it becomes very complicated to coordinate these different notification methods, since it is possible that some devices may have differing communication protocols and hence different ways of service bindings. To overcome this problem, the stuntmen community defines the abstract interface for this notification method and publishes a WiZNet view. The film director then can make use of this information and rely on a single method interface without considering the different types of peers.

A WiZNet peer can join and leave a community at any time. When joining a WiZNet community respectively a view a WiZNet peer must provide Web services that meet the requirements of the community. When a community defines certain interfaces and data fields then the WiZNet peer must assure that these interfaces are also available at the peer. Therefore, every WiZNet peer makes use of a simple matching algorithm that considers the data structure and the abstract signatures of the

Web services. Depending on the type of view the peer either stores a local copy of the view description (static view), or points to a peer that provides the view description (dynamic view).

Another possibility is the creation of communities that only allow community members to search for registry information. This allows to structure communities efficiently. Consider the example of dedicated peers that are defined as gateways to other communities. These peers allow the construction of hybrid peer to peer structures with a peer that has the ability to perform searches that are beyond the scope of the community. By the definition of “private” communities it also possible to set up test scenarios within the WiZNet network that allow only a small group of peers the access to the registry, before applying the view on a larger group of peers. This can be illustrated by the stuntmen crew of our working example. The stuntmen crew decides to build a working group with a leader that is responsible for the provided registry information of the stuntmen crew. The leader acts as a gateway to the community and is able to decide what kind of information is visible for non members. When the leader decides to join the movie view, every member of the stuntmen crew is automatically a member of the movie view.

WiZNet communities provide membership services. As soon as a WiZNet peer joins a view it becomes a member of the view. This kind of membership can be regarded as static membership because the peer decides on its own to join the view. Another possibility is the membership where a WiZNet peer is a member of the community without its knowledge. The WiZNet peer needs just to fulfill the view specification and publish the Web service. This kind of membership has a dynamic behavior because a peer is not always available and not always a member of the view. Every peer type can join a dynamic view.

## 5.2 WiZNet View Profiles

The objective of WiZNet view profiles is to describe the functionality of Web services in a given context and to provide meta data about the views. From the logical standpoint, WiZNet View profiles consist of three parts (see figure 6). 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 WiZNet service profiles that provide registry information (see section 5.3).

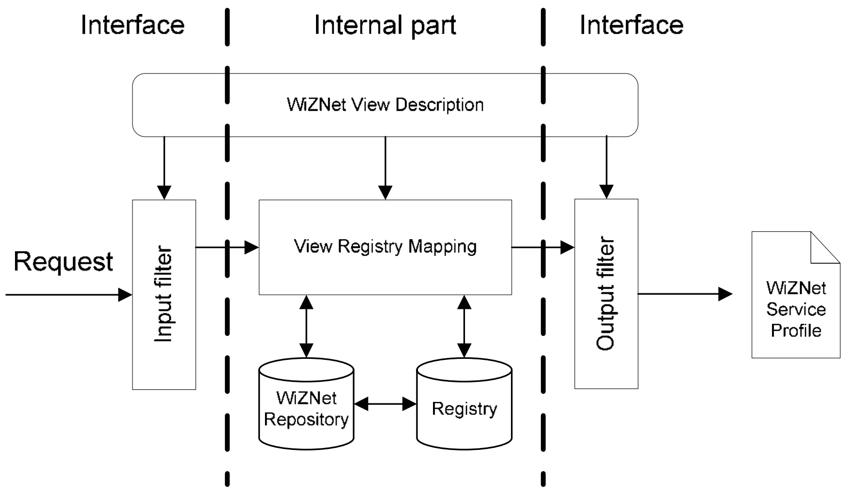

**Fig. 6.** WiZNet View overview

These three logical categories are modeled into five categories that provide view information, as shown in figure 7.

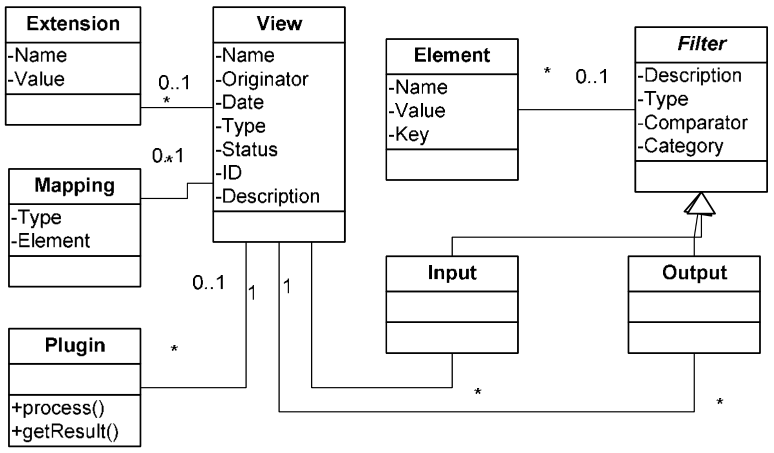

**Fig. 7.** WiZNet view profile

The `View` category 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 WiZNet 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 2 summarizes all available fields of the view category.

| Property | Description |
|---|---|
| Name | A brief, human readable name of the view |
| ID | The identification of the view |
| CreateDate | Date, when the view was created |
| ExpireDate | Date, when the view expires |
| Type | Type of the view (private or public) |
| Status | Status of the view (dynamic or static) |
| Originator | Creator of the view |
| Description | A brief, human readable description of the view. |

**Table 2.** Available fields in the View category

The `Filter` category provides interfaces for the input and output filter of views. Every filter consists of abstract method descriptions and/or attributes declarations. 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 3 summarizes the data format of filters.

| Property | Description |
|---|---|
| Description | A brief, human readable name of the filter |
| Type | Type of the filter (private or public) |
| Source Element | Element that is being compared |
| Target Element | Element that defines |
| Comparator | Comparison operator of Elements (equals, less, etc.) |
| Category | Status of the view (dynamic or static) |
| Methods | Abstract method descriptions of Output filters |

**Table 3.** Available fields in the Filter category

WiZNet filter rules are simple expressions that define relations between entities and values. Entities are names of registry entry attributes or registry entries themself. These entities are compared according to the specified operator with other entities. If the registry provides matching entries, a set of matching entities is returned, otherwise an empty set is returned. To illustrate the function of WiZNet filter rules, consider the input filter of the following example. The filter rule filters all incoming requests for the occurrence of an element with the name `Member` and the value `Movie`. Every incoming request that does not include the attribute `Member` is ignored.

```
<Filter type=”Input” category=”Property”>
  <Element type=”Source”>
    <Name>Member</Name>
  </Element>
  <Comparator>equals</Comparator>
  <Element type=”Target”>
    <Value>Movie</Value>
  </Element>
</Filter>
```

The `Mapping` category models declarative descriptions of view registry mappings. These mappings are used to transform registry information into view based Web service descriptions, respectively into WiZNet service profiles. Mappings specify element names of the registry data model and their corresponding element name in the WiZNet service profile. The mapping specification defines declarative mappings of different elements to the underlying registry data model. Table 4 summarizes the fields of the mapping category.

| **Property** | **Description** |
|---|---|
| Type | Type of the filter (private or public) |
| Element | Abstract method descriptions of Output filters |

**Table 4.** Available fields in the Mapping category

The view-registry mapping consists of two parts: (a) a generic mapping that specifies an abstract description and (b) a concrete instance based mapping of registry entries to WiZNet 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 `Registry` mapping between attributes `Name` and `Surname`. `Internal` mappings are used for the mapping of elements that are described with WiZNet service profiles. `Registry` refers to mappings of registry elements and `External` defines the mapping to external resources. `Ignore` removes the attribute `Name` and `default` assigns the attribute `name` with the value `Peter`. `Link` marks a link element. This element identifies external accessible resources and acts as entry point for external resources.

```
<Mapping type=”Internal”>
  <Element type=”Source”>
    <Name>Name</Name>
  </Element>
  <Element type=”Target”>
    <Name>Member</Name>
    <Element type=”Target”>
      <Name>Surname</Name>
    </Element>
  </Element>
</Mapping>
<Ignore>
  <Element>
    <Name>Name<Name>
  </Element>
</Ignore>
<Link>
  <Target>http://www.mymovie.org/moviedescription.xml</Target>
  <Element>Description</Element>
</Link>
<Default>
```

```
  <Element>
    <Name>Name</Name>
    <Value>Peter</Value>
  </Element>
</Default>
```

When implementing the view-registry mapping by use of plug-ins, a more flexible and powerful filtering is possible. The `Plug-in` category provides an abstract interface description that defines the interface that must be implemented by plug-ins. Plug-ins are needed for registry data transformation that cannot be described by declarative mappings, allowing a flexible and dynamic transformation of registry content. The WiZNet plug-in interface defines two methods for the operations needed by the view. A plug-in enables the view to access external resources for the transformation of the view information. The WiZNet plug in interface provides the following operations:

```
process(input d)

getResult(filter f)
```

The method `process` executes the data transformations according to the input parameter. The result is returned with the method `getResult`. This method returns the data that fits the view description in the specified format. WiZNet views can also be combined or composed in an arbitrary way. For example, the result of a view can serve as input for another view, or the results of two views are combined together for the input for another view, etc.

The `Extension` category acts as container for name-value pairs. These pairs model additional information that is not specified by the categories of WiZNet 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 copyrights statement. The example below shows the use of the extension mechanism for the example above. Table 5 summarizes the fields if the extension category.

```
<Extension type=”Instance”>
  <ID type=”WiZNet”>1234</ID>
  <Element type=”Attribute”>
    <Name>Copyright</Name>
   <Value>2004 by Peter Pan Inc.</Value>
  </Element>
<Extension>
```

| Property | Description |
|---|---|
| Type | Type of the filter (private or public) |
| Element | Abstract method descriptions of Output filters |

**Table 5.** Available fields of the Extension category

## 5.3 WiZNet Service Profiles

Although related, WiZNet service profiles and WiZNet view profiles represent different problems and require different abstractions. The objective of WiZNet service profiles is to provide a common description for Web services. WiZNet service profiles describe the functionality of Web services so that Web services can be selected by their functionality. Web services can be specified within given contexts, or in other words, Web services can be classified with the help of views.

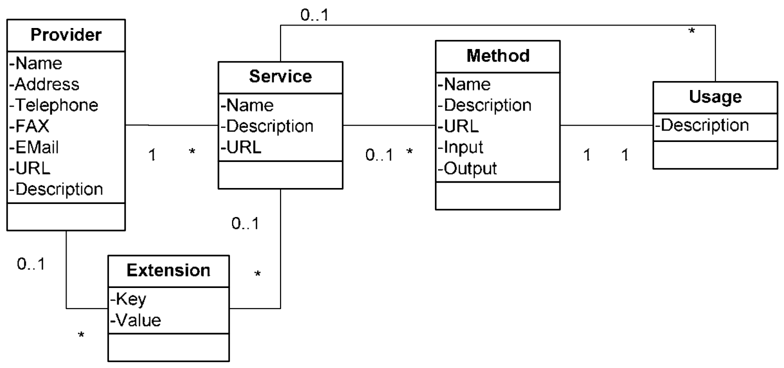

**Fig. 8.** WiZNet service profile

WiZNet service profiles consist of 5 categories, as shown in figure 8. WiZNet provides a built in service profile that acts as default view that unifies descriptions of all available Web services. In addition, the default view defines a global common context of all WiZNet peers. This context also specifies all available operations WiZNet peers provide. Every operation provided by WiZNet peers is described by WiZNet service profiles.

The `Provider` category contains information about the provider of the Web service. The provider category identifies the type of WiZNet peer and stores additional data about the Web service provider like name, address, description and URL. Table 6 summarizes the available fields of the provider category.

| 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 6.** WiZNet provider profile

The Service category 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 7).

| 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 7.** General Web service data provided by WiZNet service profiles

The `Method` category 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 (see section 5.8 for a detailed discussion of Web service method invocation). WiZNet Table 8 summarizes the available fields of the method category.

| 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 8.** Abstract method description of Web service methods

The `Usage` category is closely related to the method category. 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 category. 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 film 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 film 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. Note, that the description is similar to the actual method invocation described in section 5.8.The key difference is the additional description of the method.

```
<Usage>
  <Invocation>
    <Description>
      The hiring of film crews needs three parameters, the name of the
      Film crew and two dates. The result is OK if the film crew is
      hired or NOK if the film 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>
```

Like WiZNet view profiles, WiZNet service profiles provide an extension mechanism. Additional information about concrete Web services is modeled as name value pairs and stored in the local repository.

## 5.4 Dynamic vs. Static Views

WiZNet views define communities. Depending on the type of the view, a view can either be of dynamic or static character. A dynamic view is a loose coupled group of Web services that temporarily meet the requirements of a view description and may exist only for a certain time. Consider the working example with extras that are needed for a mass scene. The film director defines an adequate view and publishes it. Every member of the film 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. In contrast to dynamic views, a static view defines a long lasting membership. The view must be explicitly terminated by the view originator. Unless a WiZNet peer does not terminate the membership explicitly it is considered as member of the view, regardless of its actual network status. Other differences to dynamic views are the member list that is maintained by the view originator and the view replication. The member list serves as filter for the access of the Web services of a view or as notification list for changes. Consider for example a static view that defines the project team of the film. The project team is assigned to the movie for the duration of the project. The film director that is the originator of this view is able to restrict access to Web services, like planning services, location information services to the members of the project team that need this kind of services. A static view is also replicated among its members. A WiZNet peer notifies the view originator of its presence in the view.

Every public view description defines a Web service community. A Web service community can be either static or dynamic. A static Web service community is a long-term relationship among related Web services respectively Web service providers. A static community exists either for a previously defined amount of time or until a certain event takes place. Consider our working example with the relationship between the film 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. In contrast to the static view illustrates the example above a dynamic community. The membership is specified through the properties member and salary. As soon as a Web service provider is able to fulfill the specification the Web service provider becomes a member of the community. If the Web service provider does not meet the requirements it is no longer part of the community. A Web service provider may also join and leave a dynamic community at arbitrary times and later return to the community. Thus a dynamic community provides the necessary features for the building of ad-hoc communities. Note that in both cases the membership is defined by the occurrence of an event. While a static community exists until the community is explicitly canceled, a dynamic community exists only as long members are available in the community. A static community respective view also exists if no member is present.

## 5.5 Private vs. Public View Descriptions

A WiZNet view can be classified as public or private. A WiZNet peer that implements a view specification becomes automatically a member of the respective view community. Public view descriptions define a global context for Web service discovery, Web service publishing and Web service invocation. The example below shows a simple public view description that defines a filter for Web service registry entries that are members of the film view that have a salary of 10000 or more and provides two Web services. A Web service method `getDescription` that returns a description of the Web service and a method hire that is needed to hire a film crew for a given time that returns

```
<View name=”movie maker” type=”public” behavior=”dynamic”>
  <Identifier>1223</Identifier>
  <Originator>
    <Name>Peter Fando Movie Business Inc.</Name>
    <URL>www.peterfando.com</URL>
  </Originator>
  <URL>www.peterfando.com/movie.vdl</URL>
  <Description>This simple view defines a filter related to film
  making and general movie business.
  </Description>
  <Filters>
    <Filter type=”input” category=”property”>
       <Name>member<Name>
       <Comparator>equal</Comparator>
       <Value>film</Value>
    <Filter>
    <Filter type=”input” category=”property”>
       <Name>salary<Name>
       <Comparator>greater</Comparator>
       <Value>10000</Value>
    <Filter>
  </Filters>
  <Methods>
    <Method>
      <Name>getDescription</Name>
      <Parameters>
        <Parameter type=”output”>
          <Name>result</Name>
        </Parameter>
      </Parameters>
    </Method>
    <Method>
      <Name>hire</Name>
      <Parameters>
        <Parameter type=”input”>
          <Name>from</Name>
        </Parameter>
        <Parameter type=”input”>
          <Name>until</Name>
        </Parameter>
        <Parameter type=”output”>
          <Name>result</Name>
        </Parameter>
       </Parameters>
    </Method>
  </Methods>
</View>
```

Every WiZNet peer is also able to define a private view of the distributed Web service registries, independent of the membership of a community. A private view allows WiZNet peers to define private contexts for Web services. A private context enables every WiZNet peer its own specialized perspective on the distributed Web service registries. A private view always has a dynamic character. Consider the example below that specifies view with filter for the membership of the film and the stuntmen view. The output of the view consists of two properties, the name and a description of the Web service provider.

```
<View name=”stuntmen” type=”private” behavior=”dynamic”>
  <Identifier>122</Identifier>
  <Originator>
    <Name>Peter Fando Movie Business Inc.</Name>
    <URL>www.peterfando.com</URL>
  </Originator>
  <Description>This is a private view.</Description>
  <Filters>
    <Filter type=”input” category=”property”>
       <Name>member<Name>
       <Comparator>equals</Comparator>
       <Value>film</Value>
    <Filter>
    <Filter type=”input” category=”property”>
       <Name>member<Name>
       <Comparator>equals</Comparator>
       <Value>stuntmen</Value>
    <Filter>
    <Filter type=”output” category=”property”>
      <Element name=”name”>
        <Element name=”provider”/>
      </Element>
    <Filter>
    <Filter type=”output” category=”property”>
       <Name>description<Name>
    <Filter>
  </Filters>
</View>
```

The main difference between public and private views is the way the view transformations are realized. A private view acts like a local filter for data. Every request result is transformed at the peer according to the view specification. If request is executed in the context of a public view then every WiZNet peer that is a member of the view is responsible for the necessary transformations of the data. A public view also defines a membership of a community whereby a private view does not. Table 9 summarizes the features of private and public views.

| | **Private View** | **Public View** |
|---|---|---|
| Behavior | Dynamic | Dynamic, Static |
| Membership | No | Yes |

| | | |
|---|---|---|
| Notification service | No | Yes |
| Data transformation | Local | Remote, Distributed |

**Table 9.** Features of private and public views

## 5.6 View based Web service Discovery

The view based Web service Discovery puts a Web service registry entry respectively the Web service Description in a view context. Depending on the view specification, a view compliant Web service may provide an adequate Web service description, additional Web service information like quality of service attributes or ontology information. A WiZNet view unifies two aspects of a Web service description. It provides high level Web service information like Web service provider name, address, etc. and concrete information about a Web service like invocation URL or methods that can be invoked by Web service requestors. In addition Web service requestors can use view meta data for the view based discovery. View meta data provides additional information that is out of the scope of the usual Web service description. The main benefit of view information is in the Web service discovery process where a Web service requestor may use view information for an effective Web service discovery.

The view based discovery process consists of two steps that depend on the nature of the view. Consider the following example of a search request that uses the public view with the name movie maker. The example returns the salary and the address of the provider and all available Web services of the provider Peter Pan Inc. with Web service descriptions that are members of the view.

```
<Message type=”search” messageid=”100”>
  <View name=”movie maker” type=”public” behavior=”dynamic”>
    <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
  </View>
  <Filter type=”input” category=”provider”>
    <Name>name<Name>
    <Comparator>equals</Comparator>
    <Value> Peter Pan Inc.</Value>
  <Filter>
  <Filter type=”output” category =”provider”>
    <Name>name</Name>
  </Filter>
  <Filter type=”output” category =”extension”>
    <Name>salary</Name>
  </Filter>
  <Filter type=”output” category =”provider”>
    <Name>address</Name>
  </Filter>
  <Filter type=”output” category =”service”>
    <Name>description</Name>
  </Filter>
  <Filter type=”output” category =”usage”>
    <Name>usage</Name>
  </Filter>
</Message>
```

Depending on the view type the result of the search request is transformed into the expected format by the peers that provide matching Web services. The following example shows the corresponding result of the view based Web service search. Note, that the result does not contain a reference to the methods `getDescription` and `hire` because the view defines already these methods and their parameters.

```
<Message type=”search result” messageid=”100”>
  <Provider>
    <Name>Peter Pan Inc.</Name>
    <Address>Goofy Street 10</Address>
    <Salary>12000</Salary>
    <Services>
      <Service>
        <Description>
          A simple rental Service for Stuntmen equipment.
          The equipment is related to underwater stunts. This includes
          for example diving suits and huge plastic sharks.
        </Description>
        <Usage>
          <Description>
            The usage of this Service is limited to registered
            costumers only. The registration form is available at
            www.peterpan.com/register.html.
          </Description>
          <URL>www.peterpan.com/register.pdf</URL>
        </Usage>
      </Service>
    </Services>
  </Provider>
</Message>
```

In contrast to the example above, the search within private views leads to a result that corresponds to the WiZNet meta model. The following example shows the result of the search from our working example without the use of the view movie maker. The result contains the WiZNet Service and WiZNet Provider profiles.

```
<Message type=”search result” messageid=”100”>
  <Provider>
    <Name>Peter Pan Inc.</Name>
    <Address>Goofy Street 10</Address>
    <Salary>12000</Salary>
    <Services>
      <Service>
        <Description>
          A simple rental Service for Stuntmen equipment.
          The equipment is related to underwater stunts. This includes
          for example diving suits and huge plastic sharks.
        </Description>
        <Usage>
          <Description>
            The usage of this Service is limited to registered
            costumers only. The registration form is available at
            www.peterpan.com/register.html.
          </Description>
          <URL>www.peterpan.com/register.pdf</URL>
        </Usage>
        <Methods>
          <Method>
            <Name>getDescription</Name>
            <Parameters>
              <Parameter type="output">
                <Name>result</Name>
              </Parameter>
            </Parameters>
          </Method>
          <Method>
            <Name>hire</Name>
            <Parameters>
              <Parameter type="input">
                <Name>from</Name>
              </Parameter>
              <Parameter type="input">
                <Name>until</Name>
              </Parameter>
              <Parameter type="output">
                <Name>result</Name>
              </Parameter>
            </Parameters>
          </Method>
        </Methods>
      </Service>
    </Services>
  </Provider>
</Message>
```

## 5.7 View based Web service Publishing

The publishing of a Web service in a public view context adds a service to a WiZNet view. Depending on the type of view a WiZNet peer needs to perform different steps to become a member of a view. A WiZNet peer that publishes a Web service in a static view the peer first checks its Web service description for conformance with the view. If the Web service meets the requirements of the view the peer sends a Service Publishing Message (SPM) to the view originator. The SPM contains only information about the provider, because the provider publishes a Web service description in the context of the view. The view originator registers the new member of the view and returns a View Acknowledgement Message (VAM). The publishing also triggers a View Broadcast Message (VBM) to all members of the view that informs the member of the new Web service respectively the new member of the view. If a WiZNet peer joins a dynamic view the peer ensures that the Web service meets the requirements of the view. The WiZNet peer also provides a link to the original dynamic view description of the view originator.

If the WiZNet peer does not provide a view description, the Web service is published in the default view. Note, that the publishing of a Web service in the default view adds the Web service to the global context and makes it available for every WiZNet peer. In this case the Web service description fits the description provided by the WiZNet service profile (see section 7). The example below shows the publishing of a Web service description in the static view `movie` that is provided by the director and the corresponding VAM and VBM.

```
<Message type=”SPM” context=”view”>
    <View>
      <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
    </View>
    <Provider>
      <Name>Peter Pan</Name>
      <Address>Peter Pan Ave.</Address>
      <Description>Provision of magical effects</Description>
    </Provider>
  </WSP>
</Message>
```

```
<Message type=”VAM” context=”view”>
  <View>
    <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
  </View>
</Message>
```

```
<Message type=”VBM” context=”view”>
  <View>
    <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
  </View>
  <Provider>
    <Name>Peter Pan</Name>
    <Address>Peter Pan Ave.</Address>
    <Description>Provision of magical effects</Description>
  </Provider>
</Message>
```

## 5.8 View based Web service Invocation

The view based Web service invocation under a view context leads to a unified and transparent Web service access. WiZNet Views define interfaces of the provisioned Web services, specified by WiZNet service profiles. WiZNet Views and service profiles 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. WiZNet also provides direct binding-independent access to Web services. WiZNet 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 film crew members that are able to provide such a service, respectively are able to build a view conform wrapper for a salary service. A film 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. The film director sends a Web service invocation message (WSI) to the members of the stuntmen crew. The WSI includes view information and the parameter needed by the Web service. Every WiZNet peer that receives a WSI message performs the needed transformations and invokes the Web service. On completion of the Web service, every member of the stuntmen crew sends a Web service completion message (WSC) with the result back to the film director. The following examples show a WSI message and the corresponding WSC message of a film director that queries a stuntmen crew member for its salary.

```
<Message type="Web Service Invocation">
  <View>
    <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
  </View>
  <WSI>
    <Method>
      <Name>getSalary</Name>
      <Parameters/>
    </Method>
  </WSI>
</Message>
```

```
<Message type="Web Service Completion">
  <View>
    <Identifier>www.movies.org/WiZNet:director:1223</Identifier>
  </View>
  <WSC>
    <Method>
      <Name>getSarlay</Name>
      <Parameters>
        <Parameter type="output">
          <Name>result</Name>
          <Value>10000</Value>
        </Parameter>
      </Parameters>
    </Method>
  </WSC>
</Message>
```

## 5.9 View identification

Every view has its own identifier. The identifier consists of the URL of the WiZNet peer, the WiZNet peer name and number. The number is generated by a local counter that is incremented with every new view definition. The example below shows an example of a view that is located at www.movies.org/WiZNet. The WiZNet peer has the name "director" and the view is the first view that is created at this node.

```
www.movies.org/WiZNet:director:1
```

## 6 WiZNet Registry Integration Example

This section gives an example that illustrates the registry integration of UDDI and ebXML registries. Consider our working example with a private UDDI registry that provides Web services for film directors and a public ebXML based registry that provides Web services for a stuntmen crew (see figure 9).

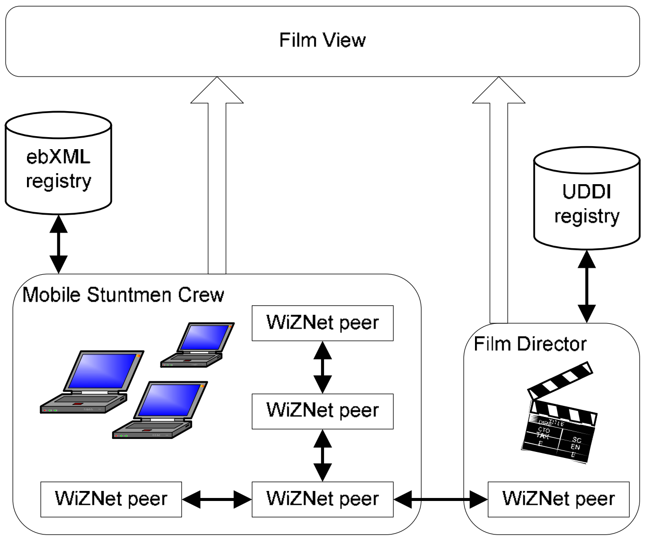

**Fig. 9.** Integration of UDDI and ebXML registries

Both registries provide differing Web service registry entries. For example, if the film director wishes to obtain information about the stuntmen crew's Web services, he must query the ebXML registry separately. The WiZNet architecture is used to integrate both registries. Every member of the stuntmen crew and the film director are connected within the WiZNet peer to peer network. We further define a view with the name `Film` that acts as common data model. It consists of Web service registry entries that are stored in these two different Web service registries. For the sake of simplicity we choose that our view provides just a basic publishing and discovery service that includes the following information about the Web services:

- Web service Name
- Web service Provider Name
- Web service Description
- Web service Invocation URL
- A single Web service method that is executed
- Web service Provider Category
- Quality of Service Description

We now focus on the mapping of the view definition on the UDDI and the ebXML data model respectively the registered Web services. Our example uses two different types of mappings. We use generic mappings to provide a general mapping for all elements of the film director’s registry respectively the stuntmen’s registry. Instance mappings are used to associate concrete instances of the Web service registry with instances of the WiZNet repository for the augmentation of additional information that is not available in the Web service registry. In our example, instance mappings are used to provide Quality of Service descriptions of the stuntmen crew’s Web services in a brief human readable form. To illustrate our approach, the following example shows a portion of a generic view to UDDI mapping. The example assumes that the Web services provide an external WDSL description file.

```
<Mapping type=”Registry”>
  <Element type=”Source”>
    <Name>Provider</Name>
    <Element>
      <Name>Name</Name>
    </Element>
  </Element>
  <Element type=”Target”>
    <Name>businessEntity</Name>
    <Element>
      <Name>Name</Name>
    </Element>
  </Element>
</Mapping>
...
<Mapping type=”External”>
  <Element type=”Source”>
    <Name>Service</Name>
    <Element>
      <Name>Method</Name>
    </Element>
  </Element>
  <Element type=”Target”>
    <Link>
      <Element type=”Source”>
        <Name>Service</Name>
        <Element>
          <Name>SpecificationLink</Name>
          <Element>
            <Name>accessURI</Name>
          </Element>
        </Element>
      </Element>
      <Element type=”Target”>
        <Name>portType</Name>
        <Element>
          <Name>operation</Name>
          <Element>
            <Name>name</Name>
          </Element>
        </Element>
      </Element>
    </Link>
  </Element>
</Mapping>
```

```
. . .
```

The example above illustrates the use of the different types of mappings. The external mapping is needed, because the actual invocation URL of the Web service is stored in the binding portion of the WSDL file [14]. The external mapping uses a nested mapping to specify the corresponding WSDL element. The mapping engine follows the link specified in the UDDI element. The rest of the mapping specifies the element of the external document that contains the needed information. The registry mapping maps the UDDI registry entries with WiZNet service profiles.

The mapping process consists of two steps. In the first step, we use generic mappings to create WiZNet service profiles from registry information. The generic mapping is used for the translation of all WiZNet requests to the underlying registry. The second step is the mapping of concrete Web services respectively their descriptions to the view. Consider for example the combination of an UDDI registry and registry entries with links to WSDL files. The generic part of the mapping specifies the UDDI entries that fit the view description. The second part, the concrete mapping of the Web service instances, returns the Web services that fit the view description and provides additional information from the repository. Our example uses human readable Quality of Service descriptions that are stored in the repository.

Figure 10 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 method of indirect retrieval is marked separately in the mapping. The mapping engine follows the link and reads the information from the specified resource. The second part of the mapping uses the key pairs to associate information of the WiZNet repository with the registry information. The following example shows a key pair mapping of UDDI registry entries with entries of the WiZNet repository. Note, that the mapping is a one-to-many mapping that allows the mapping a single Web service registry entry to a single entry and vice versa. In our example, the elements of the WiZNet repository provide information about the category of the Web service provider that is represented as short human readable text.

```
<Mapping type=”Instance”>
  <Element type=”Source”>
    <Key type=”UDDI”> uddi:FF5041D0-F5D4-11D6-82AC-000629DC0A7B</Key>
  </Element>
  <Element type=”Target”>
    <Key type=”WiZNet”>12345</Key>
  </Element>
</Mapping>
```

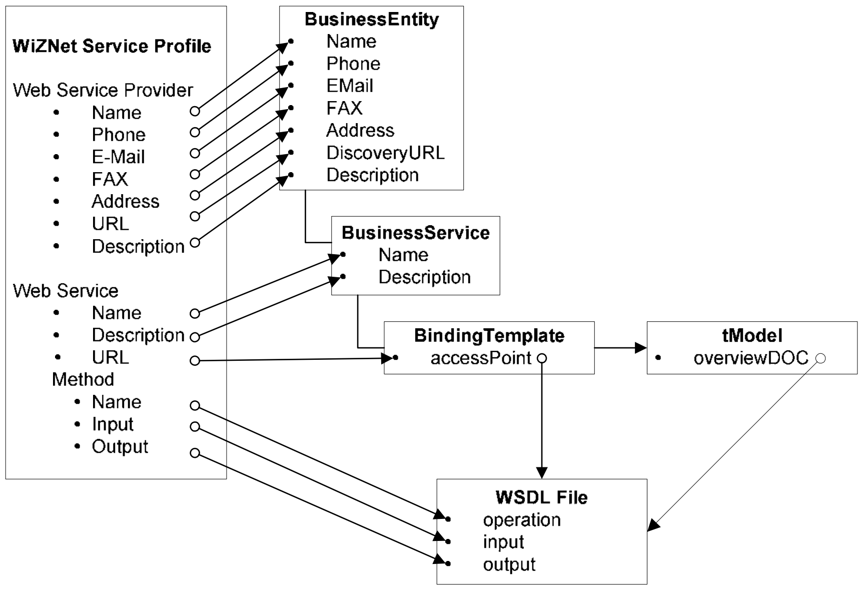

**Fig. 10.** WiZNet Service Profile to UDDI Mapping

The mapping of the WiZNet 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 11 illustrates the WiZNet Service Profile mapping of the ebXML registry model. The ebXML Organization class provides the same basic information as provided by the WiZNet Web service provider and is mapped directly on the WiZNet 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 WiZNet 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 WiZNet service profile. Note that the associations between the registry objects are not explicitly shown.

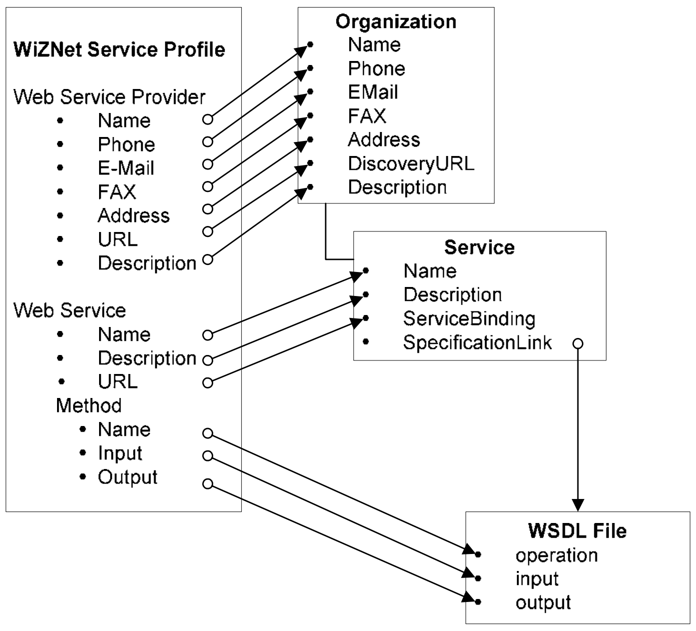

**Fig. 11.** WiZNet Service Profile to ebXML Mapping

In the case of ebXML mapping we present an example for programmatic mapping. The input of the programmatic mapping is the class `input` that specifies all fields and get and set methods of the view. The input is parsed and the corresponding field values are filled. In our example the description of the Web service is provided by an external file, the name of the Web service is stored in a relational database. Note that the example provided only generic mappings, or, in other words, transforms all registry entries to view conform WiZNet service profiles.

```
Public void process(input i) {
  // create new container for web service profile
  WSP wsp = new WSP();
  // parse the input and read all fields
  inputElements ie = getElements(i);
  for all ie {
    // check for the name of the source element
    // and read the corresponding value from the registry
    // in case of external information we access the resource and
    // look for the desired information
    if ie.type=”External” {
      resource = getfromURL(http://www.movies.org/);
      description = parse(resource, “description”);
      wsp.setDescription(description);
    }
    ...
    if ie.type=”Registry” {
      if ie.Name=”Name” {
        ODBCConnection.open("ebXML");
        name = execute("select name from service");
        wsp.setName(name);
        }
      ...
      }
    }
  }
}
```

## 7 WiZNet Services

A WiZNet peer provides a set of core services. These services include a Web service Publishing Service, a Web service Discovery Service, a WiZNet view service and a notification service. These core services are available at every type of WiZNet 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. The Web service Publishing Service provides the means for Web service publishing in the WiZNet peer network. The Web service publishing can be accomplished in two ways. A peer can either publish a single service, for example a PDA provides a local Information Service about its current user or a Web service. The WiZNet Discovery Service provides an interface with methods needed by the Web service discovery. The Web service discovery can make use of arbitrary views and filters. A search for a Web service can contain a view description or a view identifier. The WiZNet Service discovery engine supports several types of discovery approaches. The basic discovery method is by provision of the Web service name. Table 10 summarizes all basic functions of the WiZNet peers.

| **Function** | **Description** |
|---|---|
| createPrivateView(view) | Creates a private view and stores the description in the local repository of the WiZNet peer. |
| createPublicView(view, type) | Creates a public view and publishes the view description. The description is stored in the local repository. The type parameter specifies the type (static or dynamic) of the view. 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 WiZNet peer that is associated with the peer identifier. |
| getProviderProfile(id) | Returns the provider profile of the WiZNet peer that is associated with the peer identifier. |
| joinView(id) | Performs the login procedure of a static view. |
| leaveView(id) | Performs the logout procedure of a static view. |
| searchGlobal (criteria) | Searches globally for the given criteria based on the WiZNet 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. |
| subscribe(event) | Subscribes the WiZNet peer to the occurrence of a certain event. |
| Unsubscribe(event) | Removes the Subscription of WiZNet peer to the occurrence of a certain event. |
| getMethodProfiles(id) | Returns all method profiles of a given Web service. |
| getServiceUsage(id) | Returns the usage description of a Web service. |

**Table 10.** WiZNet core functions

## 7.1 Quality of Service

The WiZNet extension mechanism supports the extension of registry information with arbitrary value added information. A built in example for extended registry information are Quality of Service (QoS) attributes [10]. These attributes are stored as name/value pairs in the local repository and are mapped either to concrete registry entries (instance mapping) or to classes of registry entries (generic mapping). In either case they provide information about the services, as shown in Table 11.

| QOS Attribute | Description |
|---|---|
| Availability | Represents the probability that a service is available. |
| Response Time | Marks the round-trip time between sending a request and receiving the response. |

**Table 11.** Quantifiable Quality of Service Attributes

The WiZNet peer allows for a rating of Web service with the help of QoS attributes. The rating service compares the quantifiable QoS attributes (response time, availability, etc.) with the actual measured ones. If the expected value is exceeded by a certain amount then the QoS rating is degraded be a predefined value or factor. This information is sent to the Web service provider and the provider corrects its hard QoS attributes accordingly. Usually, a Web service provider starts with a rating of 1. This rating is corrected (either up or down) according the feedback from the clients. Positive feedback decrements the rating (minimum value is 0) and negative feedback increments the rating to a maximum value of 1. Every WiZNet peer maintains a list of attributes along with their rating and provides a rating information service that lists all available QoS attributes

## 7.2 Notification Service

The WiZNet Notification Service notifies WiZNet peers about the occurrence of events. Every WiZNet peer can register itself for events. There are two types of events, change events and notification events. Change events occur when a WiZNet peer changes a Web service description or a WiZNet view definition. In this case, a broadcast message is sent to the network an every peer that has a copy of the Web service description or a copy of the WiZNet view updates its description respectively its definition. This type of event is available only for static WiZNet peers. Dynamic WiZNet peers do not get this kind of message. A notification event occurs when a pre defined event takes place. This may be for example a deadline that is exceeded. Every type of WiZNet peer may register itself for the occurrence of such events.

## 7.3 Dynamic and Static WiZNet Registry Entries

WiZNet registry entries can either be dynamic or static. A static registry entry is a long-living registry entry that is created or deleted explicitly by a WiZNet peer or by the underlying registry. A WiZNet peer that publishes a static Web service is also a static peer that is a long term member of the network. The registry entry (i.e. the WSP) is replicated among other static peers to guarantee that the Web service description is available even when the Web service provider goes offline. If the WSP 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 static WiZNet 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 static registry entries. A dynamic registry entry is entirely specified by the WiZNet peer. A lightweight peer provides per default dynamic registry entries, the other two peer types may also provide dynamic registry entries. As soon as a lightweight peer joins the network, the registry entry is available. A dynamic 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 WiZNet core services are specified as dynamic Web services with corresponding WSPs. The only difference to common dynamic registry entries is that a core WiZNet service cannot be deleted.

## 7.4 View Information Service

The WiZNet View Information Service provides information about views. The provided information includes the view description, service invocation profiles (i.e. usage information) and a list of currently active peers. Depending on the type of view (dynamic or static) the publishing or changing of a view results in a notification message to all members of the view. If a static view is changed then all members of the view are notified of this event. Each peer updates its view specification and adapts to the new version of the view. In the dynamic case, no notification message is sent. A peer that is a dynamic member of the view may not take notice of the changes and may afterwards be no member of the view anymore. It lies in the responsibility of the peer to check for updates of the view.

### 7.5 Membership Service

The WiZNet membership Service provides the means for peers to join and leave WiZNet communities. WiZNet differentiates between two types of membership, a dynamic and a static membership. The dynamic membership is closely related to dynamic views. A peer that joins a dynamic view does so by implementing the desired view specification. The peer is then implicitly a member of the view and filters all messages regarding the view. The peer does not need to perform a login procedure. The membership implicitly terminates if a peer leaves the network without further notification. The membership service in the static case is different, because the peer must explicitly join and leave the view. In order to join a static view, a WiZNet peer must contact the view originator. The view originator stores the peer in the peer list of the view and returns the view description. The WiZNet peer stores the view description in its local repository. This replication mechanism guarantees that the view description is available as long as one of the peers is online.

### 7.6 Matching function

The matching function supports complex queries containing partial keywords and wildcards. It is possible to use regular expressions in search queries. The algorithm matches abstract method specifications in combination with wildcards regarding method names and method parameters. We consider the use of semantic meta data to be difficult since the finding of classification schemas itself is a very complex task. Since our work focuses on an environment that includes dynamic and lightweight peers, semantic meta data might prove ineffective to create classification schemas. Our method of service matching does not rely on meta data. We use a structural matching algorithm based on WiZNet service profiles that returns values in the range form 0 to 1 for a given search criteria. 0 means no match at all, 1 means the perfect match. The matching function performs the matching in two steps. The first step is to translate the search query into a registry API compatible search criteria. In most cases is it sufficient to replace all regular expressions with wildcards. In the second step the result of the query is filtered with the original regular expression in order to obtain the search result. Method specifications are matched in two separate steps. In a first step, an abstract representation is created. For example the method `Boolean hire(Date from , Date until, String name)` is being mapped onto `parameter method(parameter, parameter, parameter)`. The matching algorithm compares the abstract method signatures and proposes matching methods. The matching function provides several types of signature matches. Table 12 gives an overview of the different matching types.

| **Type** | **Description** |
|---|---|
| Plug in match | The number of the abstract parameters is smaller than the number of the parameter the method provides |
| Surplus Match | The number of the abstract parameters is higher than the number of the parameters the method provides |
| Exact match | All abstract parameter can be associated with method parameters |

**Table 12.** Different matching types

The exact match returns all methods that fit the abstract method description. This is the best possible match that can be achieved. This guarantees that a method provides at least the necessary parameter for the description. However, it is possible that the method does not fit, because the semantic of the method is different from the semantics the description expects. The plug-in match is a weaker criterion, because not all parameters of the method are matched by the abstract description. In this case it may be sufficient to fill the missing parameter with default values to use the function. The last possibility is that the abstract method provides more parameter than the underlying function. In this case, the surplus parameters may be ignored.

## 8 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 WiZNet 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 a 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 WiZNet peer to peer network. They are considered as building blocks of the WiZNet network and provide high availability and processing power.

A significant part of this paper discusses the view concept of WiZNet. 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 WiZNet 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 WiZNet service profiles that act as wrapper for existing registry entries. WiZNet 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 WiZNet 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 WiZNet 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.

## 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 University of New South Wales, Queensland University of Technology. 2003

[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.Carnegie Mellon University, Pittsburg. 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. Large Scale Distributed Information Systems (LSDIS) Lab Department of Computer Science, University of Georgia. 2003

[16] Quan Z. Sheng, Boualem Benatallah, Yan Q. Zhu, Rayan Stephan, Eileen Oi-Yan Mak. Discovering E-services Using UDDI in SELF-SERV. The University of New South Wales. 2003

[17] DAML-S Coalition. DAML-S: Web services Description for the Semantic Web. 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.