Technical University of Vienna
Information Systems Institute
Distributed Systems Group

A View Based Survey on Web services Registries

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

TUV-1841-2004-19 July 9, 2004

Web services registries are a cornerstone for the emerging service-oriented architecture and constitute a critical resource for Web services. Based on a case study we systematically illustrate and evaluate current registries and compare different approaches regarding their architectures and data models in the context of two views: the human and machine (service) based views. The human view on Web services registry architectures is illustrated with the help of a case study. The machine view on Web services registry architectures is illustrated from a software-service point of view. The data model of Web services registries is described in detail from a machine based view. The corresponding human view is described from an abstract level. Finally, the Web services publishing and discovery are compared from a human and a machine based view.

Keywords: Web services registries, Service-oriented Architecture, UDDI

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/

A View Based Survey on Web services Registries

Schahram Dustdar, Martin Treiber
Distributed Systems Group, Vienna University of Technology
{dustdar@infosys.tuwien.ac.at | e9426464@student.tuwien.ac.at}

Abstract. Web services registries are a cornerstone for the emerging service-oriented architecture and constitute a critical resource for Web services. Based on a case study we systematically illustrate and evaluate current registries and compare different approaches regarding their architectures and data models in the context of two views: the human and machine (service) based views. The human view on Web services registry architectures is illustrated with the help of a case study. The machine view on Web services registry architectures is illustrated from a software-service point of view. The data model of Web services registries is described in detail from a machine based view. The corresponding human view is described from an abstract level. Finally, the Web services publishing and discovery are compared from a human and a machine based view.

Keywords: Web services registries, Service-oriented Architecture, UDDI

1 Introduction

Web services are a new paradigm for distributed computing and are designed to enable different software systems to communicate directly with each other regardless of language or platform over the Internet. According to the W3C, Web services [2] are defined as follows: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically using HTTP with an XML serialization in conjunction with other Web-related standards." As the definition implies, Web services offer standard means for the interoperability between different distributed software systems over the Internet. The Web services paradigm allows different software systems to operate in a loosely coupled way by the help of Web services brokers, respectively Web services registries. The current Web services architecture (Figure 1) consists of three different entities: Web services provider, Web services requestor/client, and Web services registry.

The Web services provider provides Web services descriptions and publishes them using a Web services broker respectively a Web services registry. The services requestor wants to fulfil a certain task with the help of one or more Web service(s). In order to locate Web services, the Web services requestor contacts a services broker in order to search for Web services. When an adequate Web services is found the Web services requestor uses the information of the Web services broker to invoke the Web services. The services broker (registry) stores information describing Web services provided by Web services providers in a registry (repository). The Web services registry allows users to search for Web services and to publish Web services descriptions.

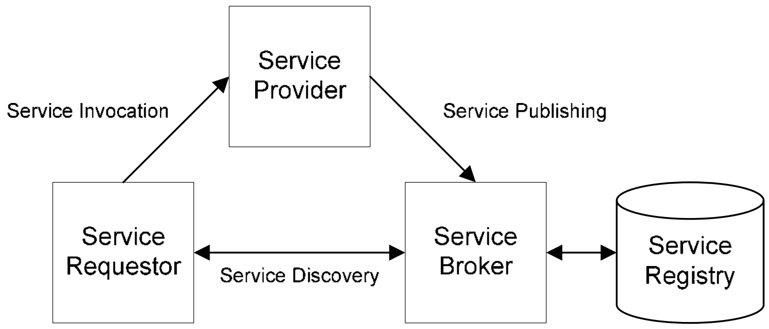

Fig. 1. Conceptual overview of Web services

This paper provides a survey of different approaches of Web services registries in both the human and the Web services context (in the rest of the paper refered to as view) regarding two dimensions: Architecture, and Data Model. The architecture describes the conceptual structure of a Web services registry in the Web services context. The data model describes the type of data and the data structure implemented by a Web services registry.

The reminder of the paper is organized as follows: Section 2 presents requirements of Web services registries and introduces the two views on Web services registries. Section 3 introduces a case study which is used throughout the paper to compare and discuss the different approaches of Web services registries regarding their data models. Section 4 gives an overview on different architectures of Web services registries, illustrates each architectural style with an example, and compares these approaches. Section 5 presents different Web services registry data models and compares the models. Section 6 illustrates how the Web services discovery is done in the different approaches to Web services registries. Section 7 presents the different Web services publishing mechanisms of the different Web services registry approaches. Section 8 concludes the paper.

2 Requirements for Web services Registries

We analyze Web services requirements based on their architectural style and their data model. The different architectural styles of Web services registries match different requirements, for example scalability, fault tolerance or maintainability, whereby each style has its own strength and weaknesses.

Another difference between Web services registries concerns the type of information stored in a Web services registry. Data about Web services can vary from basic information about a service, such as name, service classification, information about the provider, etc. to complex coordination information such as message exchange patterns, collaboration protocols or other structured information about web service capabilities.

To access stored data, a Web service offers an interface for the publishing and another interface for the discovery of Web services. The interfaces also differ between registry implementations. The requirements of Web services interfaces are closely related to requirements regarding the data model, since the Web services registry interface reflects the underlying data model. These differences also emphasize different requirements depending on the nature of the client of a Web services registry. A human Web services registry client has different requirements than, for example, an autonomous Web services registry client such as an agent.

This leads to an initial classification into "formal" and "informal" requirements regarding the information provided by the Web services registry. Formal requirements concern machine based Web services registry clients like agents or other Web services and result from the view on the Web services registry. In general, machine based clients need well structured data for the processing of Web services registry entries. Humans can deal with this type of information as well but do not depend at highly structured data.

The different views on Web services registries are depicted in Figure 2. The human view can be considered as more flexible, since humans are capable to work with (formally) structured data as well, when the amount of data is not too large. Note that Quality of Service (QoS) requirements are available in both the human and the Web services views.

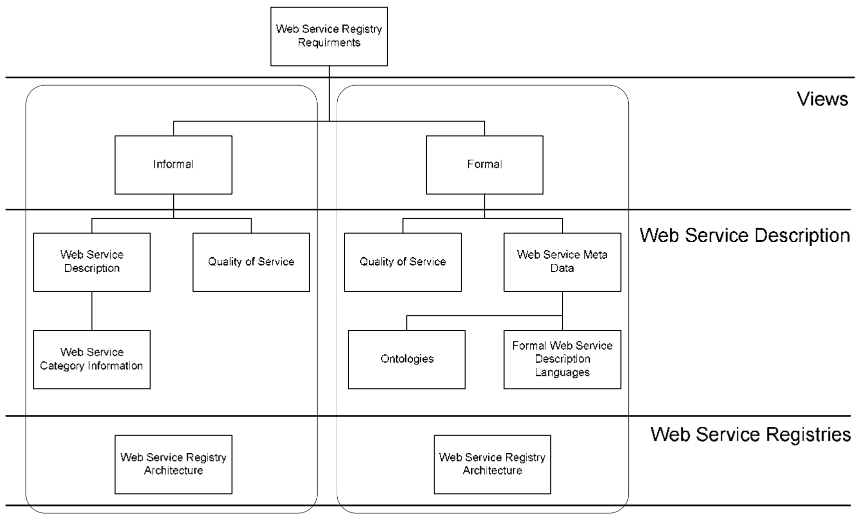
Web Service Registry Requirments
Views
Informal
Formal
Web Service Description
Web Service Description
Quality of Service
Quality of Service
Web Service Meta Data
Web Service Category Information
Ontologies
Formal Web Service Description Languages
Web Service Registries
Web Service Registry Architecture
Web Service Registry Architecture

Fig. 2. Views on Web services registries

2.1 Human View on Web services Registries

A human needs "human-readable" information about Web services. This kind of information is usually unstructured text (from a Web services view) that gives information about what a Web service generally does and information about the Web services provider, which can include name, address, and other additional information.

The provided information has no constraints regarding the type, the length of information and is rather informal. The Web services description may also provide related information about a Web services like introductionary texts about the Web services in a business context in the Web services description. With this (unstructured) information, a human is usually capable of selecting a Web service among the query results that fits the requirements of the human Web services requestor. To enable a better understanding about the Web services, categorization information can be included. From a human view, categorization information can be both, structured information in form of reference systems or informal descriptions, for example, a text containing a general business description, such as "Business activities range from the provision of stunt team equipment to expertise on physics and law".

Another important requirement regards Quality of services (QoS). The term "Quality of Service" is used to describe non-functional requirements. While it is possible to quantify some of the QoS attributes (see Table 1) other remain rather vague. These fuzzy QoS requirements can be context dependent and can be seen as guidelines. Consider for example a company which might rather use a Web service A from a company C, even if a competitor B offers a superior Web service D (lower cost, etc.), which provides the same features, due to company politics. One rationale for such a policy might be that company C is the standard Web services provider for several years, and company B is a direct market competitor). Thus, in this case, the selection of a Web services provider depends not on quantifiable attributes such as cost, performance, etc. but is based on some QoS attributes.

QoS Attribute	**Description**
Reliability	Defines the degree to which a Web services is capable maintaining the service at a given service quality.
Performance	Defines the latency and throughput of a Web services
Availability	Defines the probability of an successful Web services invocation
Security	Defines the level of security necessary to access a Web services
Cost	Defines the cost per usage of a Web services
Standards	Defines the used standards
Integrity	Defines the level of correctness regarding the execution of Web services transactions

Table 1. List of quantifiable Quality of Service attributes

An important aspect, but beyond the scope of this paper, is the human user interface for Web services registries. At first glance it may seem trivial, but there are numerous details which make the design of a human Web services interface a complex task. These details include for example skills like typewriting, the cultural background (e.g. common symbols), and graphic design rules.

2.2 Web services View on Web services registries

A machine based Web services client who queries a Web services registry for Web services depends on well structured information for the rating of other Web services. In contrast to human Web services registry clients, the Web services requestor needs explicit meta-information about the Web services when searching for Web services. This meta-information is of paramount importance when it comes to Web services comparison. Web services must be rated against another Web services to obtain an ordered query result. From a Web services view, informal descriptions are not sufficient for this kind of rating. As Figure 2 implies, the meta-information can be structured in several ways, for example, by using ontology data or other formal Web services description languages. Ontological data allows to categorize Web services and to provide a Web services requestor with similar Web services on request. Therefore, it is possible to implement automated dynamic selection policies for autonomous Web services selection. Another possibility is to establish a metric by the use of QoS attributes. Quantifiable QoS attributes allow to compare Web services and enable machine based Web services requestors to select the best fitting Web services according to their requirements.

3 Managing a film crew - a Case study

The case study presented in this section serves as a motivation for the view based comparison of the Web services registries. The example illustrates the different views on Web services registries in a *concrete* rather than in an *abstract* manner. Managing a film crew is a very complex task. There are many different types of film teams, for example, the stuntmen crew, the makeup artists, etc, which offer particular services. Some of these teams work together in a loosely coupled way providing their expertise on demand, while other teams depend on services provided by other teams and work together throughout a longer period of time. External experts offer expertise on several topics, for example physics, law, health etc. These experts are needed to make a movie reality. For example, computer scientists are needed when an actor acts as a computer expert in a movie.

A film director must be able to coordinate all these different teams and experts at different times and locations. At the same time, the film director must keep the costs as low as possible since film budgets are usually very tightly calculated. As the film director is responsible for the budget he/she has an interest in all cost-causing details of the film-making to guarantee that the film budget is not overdrawn and the movie is completed in time. To ensure the smooth and timely film-making, inter-team management is of paramount importance. The film director must enable the teams to communicate with each other in an efficient way to provide their services. Thus, the coordination of interdependent film teams is very critical for the timely completion of the movie. The different phases of the film project provide additional constraints regarding the arrangement of the film teams. During each phase a flexible configuration and composition of the different film teams is necessary. For example, when shooting an action scene the actors need stunt doubles for certain tasks (car crashes, jumps from buildings, etc.). Figure 3 shows an UML class diagram illustrating our case study:

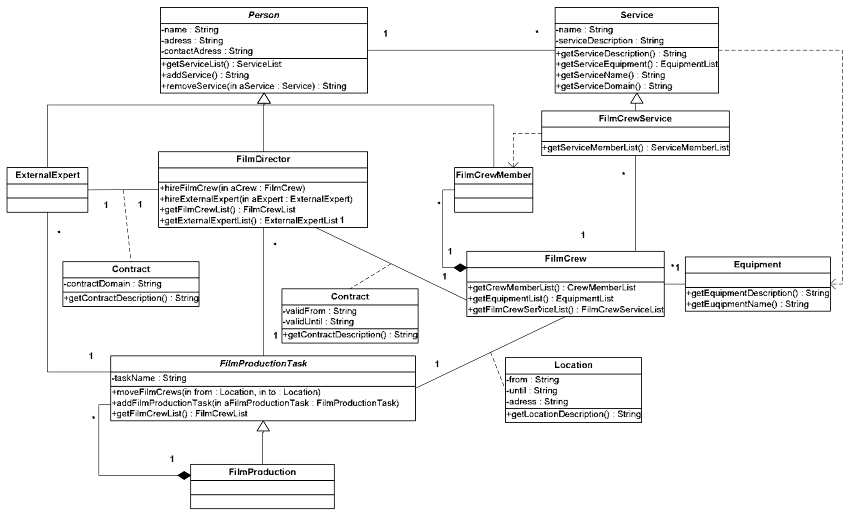
Person
-name : String
-adress : String
-contactAdress : String
+getServiceList() : ServiceList
+addService() : String
+removeService(in aService : Service) : String
Service
-name : String
-serviceDescription : String
+getServiceDescription() : String
+getServiceEquipment() : EquipmentList
+getServiceName() : String
+getServiceDomain() : String
FilmCrewService
+getServiceMemberList() : ServiceMemberList
ExternalExpert
FilmDirector
+hireFilmCrew(in aCrew : FilmCrew)
+hireExternalExpert(in aExpert : ExternalExpert)
+getFilmCrewList() : FilmCrewList
+getExternalExpertList() : ExternalExpertList
FilmCrewMember
Contract
-contractDomain : String
+getContractDescription() : String
FilmCrew
+getCrewMemberList() : CrewMemberList
+getEquipmentList() : EquipmentList
+getFilmCrewServiceList() : FilmCrewServiceList
Equipment
+getEquipmentDescription() : String
+getEuqipmentName() : String
Contract
-validFrom : String
-validUntil : String
+getContractDescription() : String
FilmProductionTask
-taskName : String
+moveFilmCrews(in from : Location, in to : Location)
+addFilmProductionTask(in aFilmProductionTask : FilmProductionTask)
+getFilmCrewList() : FilmCrewList
Location
-from : String
-until : String
-adress : String
+getLocationDescription() : String
FilmProduction
1
*
1
1
*
*
1
1
1
*
*1
1
1
1
*
1

Fig. 3. UML class diagram film working example

A film production is directed by one or more directors. Each film production consists of several sub tasks which in turn may consist of other sub tasks. Director, external expert, and crew member are all persons with particular capabilities which are provided as services. For example, a stunt man is capable of car stunts, while another stunt man is a specialist for martial arts. Film crews are hired by the director for a certain time. External experts are also hired by the director for their expertise on a particular topic. A film crew consists of one or more film crew members. Every film crew member adds its own services to the film crew. A film crew can provide film crew services which are more than the sum of the capabilities of every single crew member. For example, a car chase can be provided by a film crew rather than by a single person. Each film crew provides the equipment needed for the making of the movie. A film needs one or more film crews. Each film crew is assigned to a film production task by the location where the film crew is needed. For example a camera crew provides specialized camera equipment for the shooting of film scenes under water. Services require equipment for their realization.

4 Web services registry Architectures

Web services can be classified by their architectural style. The architectural style of Web services defines how a Web services registry is actually implemented. The implementation of Web services influences the message interaction schema between Web services registry, Web services provider, and Web services requestor. Generally, Web services registries can be classified with regard to their architecture: (a) Centralized, (b) Federated, or (c) Decentralized.

Each of these different architectural styles provides certain strengths and weaknesses regarding scalability, fault-tolerance, administrative overhead, complexity, and performance. Using our case study a human organization analogy to the particular Web services registry architecture is presented in the following sections. The human analogy shows the human view on the illustrated Web services registry Architecture. The Web services view is not explicitly stated, since the description of the directory architecture is already the technical respectively the machine view.

4.1 Centralized Architecture

In a centralized approach a single entity contains all Web services registry entries, referred to as Web services registry (Web services broker). Each Web services provider uses the central Web services registry for the publishing of its service descriptions. The Web services broker stores registry information in a central "well known" registry. Services requestors contact the service broker in order to obtain information about Web services. This model follows a traditional client/server approach where the Web services registry acts as server, the Web services provider as content producing client and the Web services requestor as an information seeking client. The publishing of a Web services involves the following four steps:

1. A Web services provider contacts the Web services registry and registers a services
2. A Web services requestor searches the Web services registry and obtains information about a Web services
3. The Web services requestor contacts the Web services Provider and obtains detailed information, necessary for the binding of the Web services
4. The Web services requestor invokes the Web services

The human analogy to a centralized Web services architecture is as follows. In our case study the film director class plays the role of the Web services registry. The film director possesses all information for the shooting of the movie and is the central coordinator of the movie. Whenever needed, the film director must be contacted. For example, when a stunt team needs make up artists the film director provides the stunt team with the necessary information about the make up artists and allows the stunt team to coordinate their activities with the make up artists.

The SELF-SERV project [4] is an example for using a centralized UDDI [1, 6, 7] based registry (Figure 4). The Services Manager component consists of three modules, namely the Services Discovery Engine [3], the Services Editor and the Services Deployer.

The Services Discovery Engine manages the registration and the location of services. The Services Discovery Engine is implemented in Java using UDDI, WSDL [19] and SOAP [18] technology. Before a service is registered in the UDDI registry, it must generate a WSDL Description and deploy the description at a public location, identified by an URI. The publishing is completed by sending a SOAP message with the Web services information to the services Discovery Engine that stores the data into the UDDI registry and makes it available in the services pool for later discovery by Web services requestors.

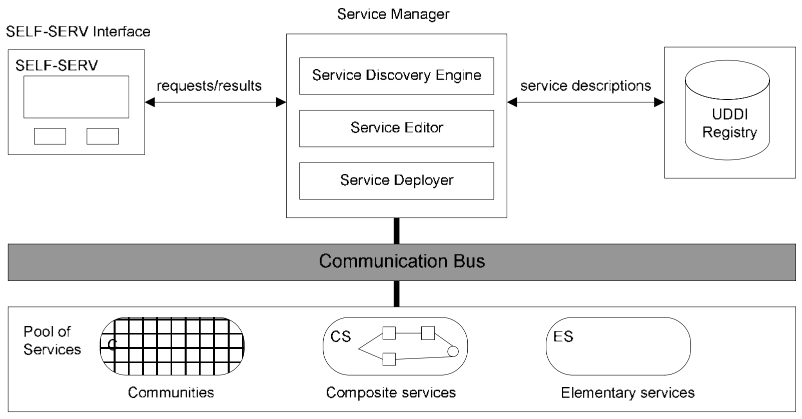

Fig. 4. SELF-SERV and UDDI

When applying our case study to SELF-SERV, each person (film director, external expert, and film crew member) publishes the service descriptions in the UDDI registry. SELF-SERV organizes the published services in service communities, composite services, and elementary services. For example the stuntmen community acts as container for services provided by stunt men crews. The stuntmen community provides descriptions of the associated services without referring to the actual stunt men service provider. This enables a director to dynamically select a service from the stuntmen pool. Film crews provide composite services that consist of elementary film crew member services. The director has the opportunity to set up composite Web services for related tasks, for instance to coordinate the stunt men with the external expert on physics.

4.2 Federated Architecture

The federated approach distributes Web services registration information among different entities in a peer to peer fashion. Dedicated nodes of the network often referenced as super peers or peer registries, store Web services registry data. This approach, sometimes called a hybrid peer to peer network, unifies aspects of centralized and decentralized Web services registries.

The registry peers provides transparent registry access through several gateways, respectively registry peers. In this mode both, the Web services provider and Web services requestor, act as in a centralized Web services environment since the distributed nature of the Web services registry is not visible for the Web services provider and Web services requestor. The process of registering and discovery of Web services is similar to the approach taken in a centralized architecture. The only difference lies in the communication overhead between registry peers when a search is performed which includes several distributed registries. The registry peers can also provide semi-transparent registry access. A Web services requestor is enabled either to make the choice between a local search in the registry of the Web services registry peer or a global search involving every registry peer of the network. The semi-transparent approach allows for specialized registries. Each Web services registry peer provides a registry which is specialized at a certain type of Web services. A Web services provider can publish a Web services in a specialized Web services registry using meta-information of the Web services registry peer about the type of Web services that are stored in the Web services registry of the peer. Form a human point of view, the federated Web services registry architecture resembles a workgroup organization. Each workgroup consists of a group of people which are lead by a group manager.

In our case study several film directors manage their groups in an autonomous way. Each film director is responsible for a certain area of the film-making. For example, a film director manages the stunt teams, while another film director manages the special effect crews. When, for example, a stunt team needs the assistance of the special effect team, it contacts the stunt team director. The stunt team director asks the special effect team director who is able to provide the needed expertise to the stunt team. [14] introduces the concept of service-syndications, where related business form groups of interest with their own UDDI peer registries that operate in a decentralized fashion. These so called super peers store a sub directory of a UDDI business registry where every syndication peer publishes its service description (Figure 5).

The super peer manages the communication between different peers and is responsible for the joining and leaving of peers of service syndications. The key concept of the service syndication is the event notification, which allows peers to operate in an independent way. Each peer can register itself for certain occurrences of events. The registry peer informs the registered peer when it obtains a matching subscription from another peer. This enables peers to form their own so called peer acquaintance group (PAG). Each PAG consists of peers having the same interests, where each peer knows every member of the PAG. The members of the PAG cooperate by propagating Web services requests to peers within their own PAG without the help of the super or registry peer.

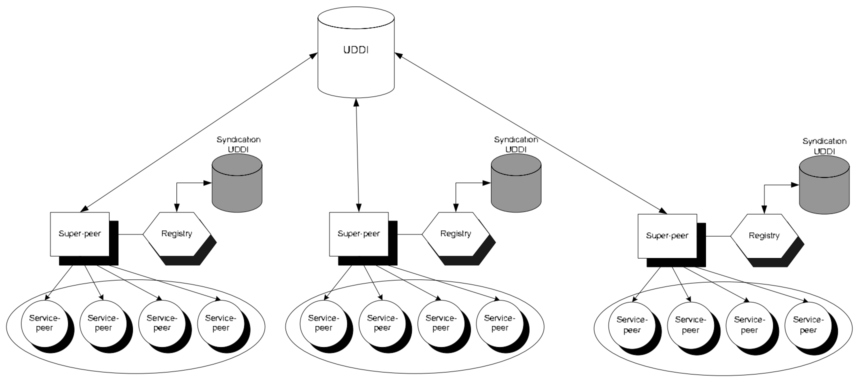

Fig. 5. Web services Syndication overview

In our case study, a service syndication maps to a community of different service providers. For example, the stuntmen service syndication offers services related to stuntmen. External experts can be grouped into an expert syndication. Each super peer of the service syndication group can be considered as sub director for the respective area. The METEOR-S [12] project implements a distributed registry structure. The system architecture consists of four layers (Figure 6). The Data Layer is responsible for the Web services registry. Each peer provides its own local registry in based on UDDI. The Semantic Specifications Layer enables the use of semantic enriched metadata. Semantic metadata is used on the Data Layer and the Operator services Layer with the help of ontologies. On the Data Layer a specialized ontology, the registry Ontology maps each registry to a certain domain. This enables registries to be grouped according their domain.

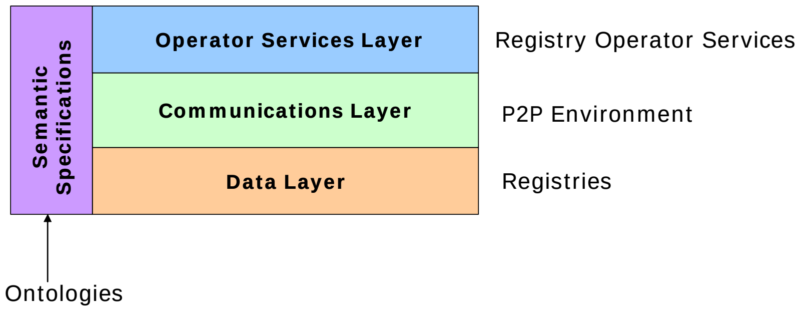

Fig. 6. METEOR architecture

The Communication Layer provides the means for communication between peers the different peers. METEOR support four different types of peers: gateway, operator, auxiliary, and client peer. Each operator peer controls a local registry and provides operator services. The operator peer provides advanced Web services discovery mechanisms based on ontological information.

Figure 7 provides an overview on the METEOR communication layers. The gateway peer (GWP) manages the access to the peer to peer network for new registry operations. The GWP is a central entity in the peer to peer network which plays the role of an entry point for registries when joining the MWSDI. The gateway peer also informs the other peers of the network as soon as updates of the registries ontologies are necessary. The operator services layer provides value added service like the semantic discovery of Web services. The operator service layer allows client peers to communicate with the registries and abstracts users from the semantic details of the Data Layer.

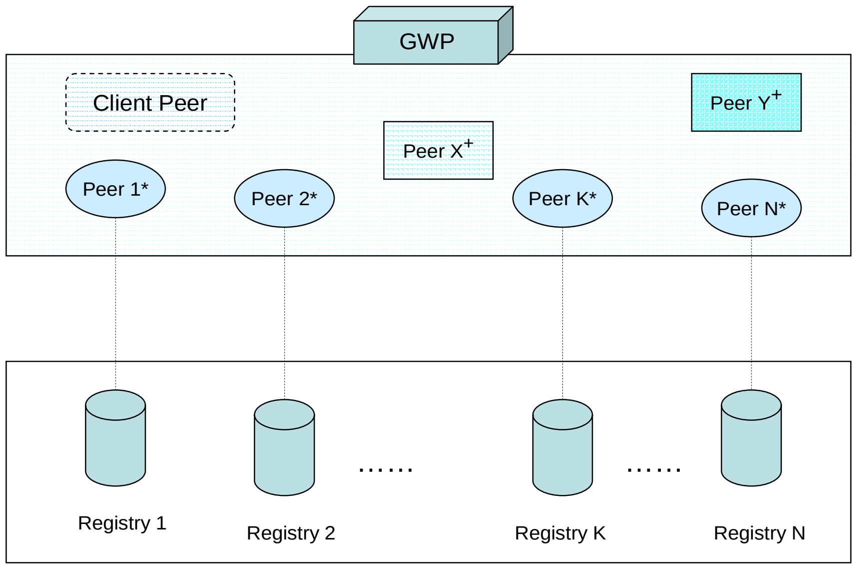

Fig. 7. METEOR communication layer overview

Because of the gateway peer, METEOR can be classified as an hybrid peer to peer network. The gateway peer may act as single point of failure, but METEOR is also capable of operation without the gateway peer. When the gateway peer fails, not all features of the Communication Layer are available, for example, it is not possible for new registries to join the network.

Now we apply our case study on the METEOR-S architecture. The film production itself is considered as the peer to peer network. The director is mapped onto the Gateway peer and is responsible for the joining (hiring) of different film crews. Once the film crews are members of the peer network they operate in a rather independent way. When, for example, a stuntmen crew wants to offer services and provides ontological data, it must contact the director first. The director informs the other peers of the new ontology and joins the stuntmen crew to the network. Crew interrelated service coordination is carried out by the different crews themselves. They need to coordinate their activities among each other without the involvement of the director. The director just gives instructions and additional information (for example a time frame in which the tasks have to be completed) and is informed about the completion if the tasks.

4.3 Decentralized Architecture

A decentralized approach implements a pure peer to peer architecture. From a functional point of view, each service provider has a local registry and acts as service provider and as service registry (broker) at the same time. The different roles are carried out by the same provider. Web services registry entries exist only as long as the Web services provider is part of the peer to peer network. As soon the Web services provider leaves the network, the registry entry is not valid anymore, since the Web services registry entry is not available any more. This implies a dynamic registry structure where the lifespan of a registry entry is limited by the connection time to a peer network. In contrast to the other Web services architectures the services publishing/discovery, and invocation requires three steps:

1. The Web services provider connects to a peer to peer network
2. The Web services requestor searches the peer to peer network for a Web services
3. The Web services requestor invokes the Web services

The distributed concept of registries applied to our case study, leads to a flat organization where no hierarchy exists. A director is like every other member of the film team. When a film crew needs the expertise of another film crew, it sends a (kind of) broadcast message to the film crews. On receiving a request, each film crew checks, if the request can be fulfilled or not. When a request can be met, the film crew contacts the requesting film crew and their activities can be coordinated. [24] presents a peer to peer registry architecture based on distributed hash tables. Web services registry information is distributed over a peer to peer network using an indexing system that is based on the CHORD [32] data lookup protocol. In this system, Web services are indexed using those keywords that describe the given Web services. Each data element is associated with a sequence of keywords that define a mapping into a multidimensional keyword space. The n-dimensional keyword space is mapped to 1-dimensional index space which is mapped onto an overlay network of peers. When a node joins the network it must know at least one node already in the network. The joining sends a join message which is routed across the network and is then inserted into the network structure.

The modeling of our case study on this architecture leads to a flat hierarchy. Each member of the network provides information about the service offerings itself. The film director and the different film crews are connected in a peer network. The director contacts different peers directly when a service is needed. The same holds true for each member of the film crew. When for example a stuntmen crew needs the expertise of external experts, it contacts the expert directly without the involvement of the director. Thus the responsibility is divided in equal shares among the film crew members. The most important aspect is the independent inter team coordination of their activities

4.4 Summary on the Architectural Styles

The human and machine views on Web services registry architectures overlap. There is no notable difference in the view, when comparing the Web services registry architectures. The concept of architectures can be applied directly to human organizational structures. A central registry offers simplified administration, since there is a single entity which has to be administrated. Furthermore there are no coordination or replication activities between different Web services registries, which add administrative overhead. At the same time this benefit is also the main drawback. A centralized Web services registry acts as a single point of failure. When the central Web services fails, it is not possible for clients to search or to register a Web services. Another problem is the limited scalability of centralized Web services registries. As the number of registry entries grows, the time for the discovery of Web services increases and also the potential hits increase since it is likely that many Web services offers a certain service. To solve the problem of limited scalability and fault tolerance, a replication schema can be implemented, where several servers offer a replicated registry. The replication of Web services registries weakens the main benefits of a centralized Web services registry, since replication needs administrative overhead to manage replicated registries at different locations. The federation of registries offers a more scaleable solution, where a peer to peer network of registry peers maintains the registry entries. The load of Web services registries can be distributed among several peers leading to increased performance when the Web services registry grows.

Another possibility is the specialization of registry peers. Registry peers can specialize on certain types of Web services. Therefore, it is possible for registries to act as market places where related businesses publish their Web services. Furthermore, it allows a registry to be smaller and more efficient regarding search times, compared to a centralized approach. Along with the specialization comes the possibility to adapt the implemented data model for specialized registries in a flexible manner. A specialized registry could offer additional registry information for Web services like quality of service information.

Federated registries are more fault tolerant because the failure of a registry peer only affects a part of the network. To ensure better fault tolerance, different registry peers can hold replicas of other registry peers, since the amount of data is less, compared to a central registry. Compared with a central approach, a federated registry has more message overhead. Global search queries need to be forwarded from registry peer to registry peer in order to carry out a global search operation. This leads to more messages in the network since the search query must be sent to all registry peers and afterwards query results from all registry peers must be sent back to the query originator from throughout the network.

The fully decentralized registry provides the best fault tolerance, because the failure of a peer does not affect any other peer, because each peer acts as a registry node itself. Another benefit is the location transparent registry, due to the fully decentralized registry structure. A Web services provider needs no knowledge about a central registry or registry peers. It suffices to know an arbitrary peer of the network to be able to publish a Web services.

The distributed Web services registry approach provides the largest flexibility, because it can evolve into any other registry architecture. It is possible to set up a federated structure where related businesses can publish their Web services in a clustered way. Another possibility is to build a single Web services registry service which acts as central Web services registry within the peer to peer network. Another benefit of distributed registries is the way they handle dynamic registry entries. A Web services can dynamically join and leave a peer network without any administrative overhead. There is no need to contact a central entity when a Web services is being published or removed from the network. Due to the dynamic nature of the distributed registry it is not possible to ensure that a registry entry exists over a certain time. In contrast, the centralized and decentralized registry solutions can guarantee the existence of Web services registry entries as long as the registries are operational. A drawback of a distributed solution is the amount of messages that circulate through the network when a search query is executed. Potentially the entire network is searched for the requested Web services. Frequent search queries can lead to a degeneration of the response time when searching for a Web services since the network bandwidth is consumed by the search messages. Table 2 summarizes the features of the three different approaches.

	Centralized	**Federative**	**Decentralized**
Scalability	Low	High	High
Fault-tolerance	None	Yes	Yes
Extensibility	No	Medium	High
Registry location transparence	No	No	Yes
Administration	Simple	Medium	Simple
Dynamic registry Entries	No	No	Yes
Flexibility	Low	Low	High
Message Overhead	Low	Medium	High
Specialized registries	No	Yes	Yes
Reliable registry Entries	Yes	Yes	No

Table 2. Overview of registry features

5 Web services Registries Data Models

Web services registries implement different data models to store registry information. Persisted data differ from simple informal Web services descriptions and formal ontological structured information. The following section describes UDDI, ebXML, WSDA, and WSIL regarding their data model and compares the different approaches from a Web services view. We use our case study throughout the section to illustrate the differences between the data models. Finally, we present a description of the human view of the data models and an analysis how the human requirements are met by the different data models.

5.1 UDDI

UDDI (Universal Description, Discovery and Integration) is a standard which is part of the Web services architecture. UDDI contains a framework for both, the specification of Web services and the specification of businesses. UDDI uses standard technologies (SOAP, XML [25], HTTP [21], TCP/IP) and is set on top of an interoperating stack. UDDI focuses mainly on the discovery of services. Web services descriptions are not part of the UDDI specification. Service descriptions like WSDL can be referenced by UDDI registry entries using tModels [23].

5.1.1 UDDI Data Model

The UDDI data model [6] is a hierarchically-structured data model. It provides a "top-down" approach, where information about a Web services is divided into several categories and each category offers more detailed information about the registered Web services. Each entity in the data model is identified by a unique universal identifier (UUID). Generally, the UDDI data model can be divided into three main categories:

- White pages
- Yellow pages
- Green pages

White pages provide general information about a Web services provider, for example business name, business description, contact information, address or phone numbers. Yellow pages provide classification data for either the company or the offered Web services. For example, this data may include industry, product, or geographic codes based on standard taxonomies. Green pages provide technical information about a Web services. This type of information includes a pointer to an external specification and an address for invoking the web service. These three categories are modeled into five in five distinct data structures:

- businessEntity
- businessservice
- bindingTemplate
- tModel
- publisherAssertion

To illustrate the function of the different data structures, each of the data structures is associated with a part of our case study. Note, that UDDI provides no direct information about the collaboration of the different member of the movie making process.

The businessEntity encapsulates information about a business or an entity that publishes information about Web services offerings. BusinessEntities include information about their name, description, services offered, and contact information. In addition to basic business information, a businessEntity can contain elements with additional information about business identifiers and business categories. Business identifiers can be arbitrary unique business identifiers and are stored in an identifier bag. Business classifications are stored in category bags. UDDI offers three built in global classification schemes, based on following standards:

- The North American Industry Classification System (NAICS) taxonomy
- The Universal Standard Products and services Code System (UNSPSC) taxonomy
- The International Organization for Standardization Geographic taxonomy (ISO 3166)

The following example illustrates a businessEnity structure for our case study (section 2) with name, contact, identifier and category information on the film director:

```
<businessEntity businessKey="A687FG00-56NM-EFT1-3456-098765432124">
  <name>Martins Movie Director services</name>
  <description xml:lang="en">
 The MMDS offers a variety of Web services for the management of
movies. The services include the selection of different film crews,
like stuntmen, makeup artists and all other film related personal.
  </description>
<contacts>
  <contact useType="US general">
    <personName> Martin Marty</personName>
    <phone>1 800 CALL MMDS</phone>
    <email useType="">office@mmds.org</email>
    <address>
      <addressLine>MMDS</addressLine>
      <addressLine>1000 Bollywood Avenue</addressLine>
      <addressLine>Bombay 1000</addressLine>
    </address>
  </contact>
</contacts>
<identifierBag>
<keyedReference tModelKey="uuid:8609c81e-ee1f-4d5a-b202-3eb13ad01823"
keyName="D-U-N-S" keyValue="01-234-345667" />
  </identifierBag>
<categoryBag>
  <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-
04411D14E384" keyName="Movie services" keyValue="123456"/>
</categoryBag>
</businessEntity>
```

The businessservice describes services offerings in a more detailed way. Every businessEntity offers one or more services, which are grouped together in the businessservice structure. The published information is similar to those of the businessEntity, including information about service name, service description, a unique service identifier and bindingTemplates related to a businessservice. The bindingTemplate acts as a container for technical information of services. This element contains information that is needed for the communication with a given service, including unique identifier, the access point of the service (for example an URL) and references to tModels. The example below shows a businessservice element including a bindingTemplate element and tMoldelInstanceDetail element using our case study. The example provides an additional description and an URL which specifies a SOAP binding for the hiring of film crews by a film director:

```
<businessservice serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
businessKey=" A687FG00-56NM-EFT1-3456-098765432124">
  <name>MMDS film crew Management</name>
  <description xml:lang="en">Hiring of film crew provided by
MMDS</description>
   <bindingTemplates>
     <bindingTemplate serviceKey="d5921160-3e16-11d5-98bf-
002035229c64"
        bindingKey="d594a970-3e16-11d5-98bf-002035229c64">
       <description xml:lang="en">
         SOAP binding for the hiring of film crews
       </description>
       <accessPoint URLType="http">
         http://www.mmds.org:8080/hire
       </accessPoint>
       <tModelInstanceDetails>
         <tModelInstanceInfo tModelKey="uuid:0e727db0-3e14-11d5-98bf-
002035229c64" />
       </tModelInstanceDetails>
     </bindingTemplate>
   </bindingTemplates>
</businessservice>
```

The main use of the tModel structure is to represent arbitrary technical details of a service. The tModel element provides pointers to external technical documents. In fact, every link to external information is represented by tModels, for example, the identifierBag uses a tModel to point to a previously registered tModel instance, representing a business identification system. The example shows a tModel element pointing to a URL that contains a WDSL specification of the hire film crew method of our case study. Additional information about the type of the specification is encapsulated in the <categoryBag> Tag.

```
   <tModel tModelKey=" uuid:0e727db0-3e14-11d5-98bf-002035229c64">
     <name>uddi-org:inquiry</name>
     <description xml:lang="en"> WSDL Document for the hire film crew
API </description>
     <overviewDoc>
       <description xml:lang="en">
         This tModel defines the API calls for hiring a film crew
       </description>
       <overviewURL>
         http://www.mmds.org/wsdl/hire.wsdl
       </overviewURL>
     </overviewDoc>
     <categoryBag>
       <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-
39B756E62AB4"
```

```
         keyName="types"
         keyValue="specification"/>
       <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-
39B756E62AB4"
         keyName="types"
         keyValue="xmlSpec"/>
       <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-
39B756E62AB4"
         keyName="types"
         keyValue="soapSpec"/>
       <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-
39B756E62AB4"
         keyName="types"
         keyValue="wsdlSpec"/>
      </categoryBag>
   </tModel>
```

The publisherAssertion element models information about different related businesses. The example illustrates a parent-child relation between two businesses, which are both identified by their UUID.

```
<publisherAssertion>
  <fromKey>0e727db0-3e14-11d5-98bf-002035229c64</fromKey>
  <toKey>0e727db0-3e14-11d5-98bf-002035229c64</toKey>
  <keyedReference tModelKey=" uuid:C1ACF26D-9672-4404-9D70-
39B756E62AB4"    keyName="MMDS International"  keyValue="parent-
child"/>
</publisherAssertion>
```

5.2 ebXML

The ebXML (electronic business XML) standard [10] defines a framework that aims to allow different businesses to find each other and to conduct business activities. ebXML specifies several interrelated components for business activities and provides a central registry or repository for storing information. In detail the ebXML architecture consists of the following components:

- Business Process Models [26]
- Messaging services [31]
- Collaborative Protocol Profiles (CPP) [27]
- Collaboration Protocol Agreement (CPA) [28]
- Registry and Repository [9, 11]

Business Process Models describes the way businesses conduct their business processes. The CPP provides general business information like name, contact, etc. Additional technical information like interface description and message requirements are also specified by CPPs. The CPA provides a negotiation between business partners for business activities. The CPA includes data about agreed service requirements upon all participating business partners. The Messaging services specify a communication-protocol agnostic method for the exchange of business messages.

The ebXML registry acts as a database for data regarding business to business communication. It follows a similar concept like UDDI registries, but is broader in scope. An ebXML registry is capable of storing arbitrary data, for example, XML schema and documents, process descriptions, Web services, ebXML CPP, ebXML CPA, context descriptions, and UML models or information about parties or even software components. The ebXML registry architecture follows a centralized approach. Businesses publish their Web services descriptions in a well known registry, which can be browsed by clients. The ebXML registry offers two separate interfaces [9]:

- LifeCycleManager Interface
- QueryManager Interface

The Lifecycle Management interface is a sub-service of the registry service. It provides the functionality required by clients to manage the lifecycle of repository items (e.g. XML documents required for ebXML business processes). The Lifecycle Manager controls the status and changes of objects during their existence in a registry. The LifecycleManager handles the submission of objects, the classification schemes of object and the removal of obsolete objects from the registry. The QueryManager interface enables clients the discovery of Web services. It provides the functionality required by clients to locate Web services. The QueryManager interface consists of two parts allowing search with SQL expressions and Filter expressions respectively.

5.2.1 ebXML registry Data Model

The ebXML registry data model provides metadata about registry items and is organized into 17 classes. This data model is capable to store arbitrary objects, making use of a build in extension mechanism. Furthermore, the ebXML data model offers a classification mechanism and allows related registry entries to be organized in packages. In detail, the ebXML data model is organized as follows: The class Association defines the relationship between a registry entry and other objects providing binary relations between objects. Associations also provide an important structural element within the ebXML registry. They provide the mean to structure the content of an ebXML registry in a hierarchical fashion. The Auditable event class is needed to generate an audit trail for the registry entry. This structure enables to protocol tracking content associated with registered users. The Classification class categorizes registry entries. This class provides the basic means for classification systems based on industries or markets. The Classification node class defines a branch in the tree structure for the classification system. The class External identifier provides a mean to identify a registered item with external keys, for example UCC code. The class External link provides a way for an object to reference Internet resources outside the registry. This may include for example a schema data with an URN that refers to another schema. The class Organization defines the submitting organization for the registry entry. It is possible to store   references to the parent organization of the submitting organization. The class Package allows for grouping registry entries together for managing the group. The class Slot provides a dynamic way to add arbitrary attributes to registry entries on base of name/value pairs. This enables extensibility within the ebXML registry data model. The following example shows our case study from section 2 applied to the ebXML data model. It implements a service description and points to a WSDL description of the hire method of the director.

```
<service id="MMDS Inc">
    <Name>
       <LocalizedString lang="en_US" value = "Martins Movie Director
services"/>
    </Name>
  <Description>
     <LocalizedString lang="en_US" value = "The MMDS offers a variety
of Web services for the management of movies. The services include the
selection of different film crews, like stuntmen, makeup artists and
all other film related personal."/>
    </Description>
  <Slot name = 'HTTP or SOAP'>
  <ValueList>
  <Value>SOAP</Value>
  </ValueList>
  </Slot>
    <serviceBinding accessURI="http://www.mmms.org/hire ">
        <SpecificationLink
specificationObject="wsdlForhirefilmcrewDescription ">
      <UsageDescription>
        <LocalizedString lang="en_US" value = "WSDL Document for the
hire film crew API"/>
      </UsageDescription>
    </SpecificationLink>
    </serviceBinding>
     </service>
  <ExtrinsicObject id="wsdlForhirefilmcrewDescription"
mimeType="text/xml">
   <Name>
    <LocalizedString lang="en_US" value="WSDL Document for the hire
film crew API"/>
    </Name>
  </ExtrinsicObject>
```

5.3 WSDA

The Web services Discovery Architecture [8] provides a Web services discovery layer on top of a grid based architecture. The discovery layer defines four interfaces along with a tuple based universal data model which enables to store arbitrary content. The interfaces in WSDA are:

- Presenter
- Consumer
- MinQuery
- XQuery

The Presenter interface enables the retrieval of service descriptions by use of HTTP(S) Get requests. The Consumer interface provides the possibility to publish content to a consumer, for example, a registry service. The MinQuery interface provides basic query support using "select-all style" queries. It provides clients with tuples in their original input format, that is, each tuple is presented in the same way as it was published before. The XQuery interface provides XQuery support. The XQuery interface allows clients more expressive search queries than the MinQuery interface. For example, it is possible to specify path expressions for hierarchical navigation. Each peer in the Web services Discovery Architecture can implement a subset the specified interfaces, depending on the role of the peer. A registry peer may implement all four interfaces, while a peer that only publishes data implements just the Presenter interface. The registry model follows one of three approaches:

- The Pull registry
- The Push registry
- A Hybrid registry

In the Pull registry approach, a content provider publishes a content link. The registry pulls the content using the content link into the registry. As soon as a content provider changes, it notifies the registry. The registry can then decide if and when the new content is pulled into the registry. In the Push registry approach, a content provider pushes both, the content link and the content into the registry. Every modification of content leads to a push of the current content to the registry. The hybrid approach implements a pull as well as a push registry at the same time.

5.3.1 WSDA Data Model

The WSDA data model specifies a unified data model based on tuples. Each tuple can be viewed as container for arbitrary data with the following data fields:

- Link
- Type
- Context
- Timestamps
- Metadata
- Content

The Link is an HTTP(S) URL and points to the content provided by the content provider. The Type describes the kind of content that is being published. The Context describes the reason why content is published or how it should be used. The Timestamps TS1, TS2, TS3, and TC provide information about modification time of a tuple and the validity of the tuple content. The Metadata element offers additional information. An example for metadata is a Web services Inspection Language (WSIL) document. The content itself can be of arbitrary nature. The retrieval is done by use of the link specified in the Link attribute. The registry entries are maintained by soft state data container to support dynamic changing of registry entries. Each data tuple possesses four timestamps which contain information about the modification time and the lifespan of the tuple. After publishing a tuple into the tuple space, a tuple is valid for a certain time. When the publisher of the tuple refreshes the lease timely the tuple stays in the tuple space otherwise it is removed. A registry in the WSDA architecture is merely an indexing service. Each registry entry points to the external description of the Web service. Our case study in section 2 is applied as follows. Using the content portion of the tuple a WSDL document is embedded. The director provides a WSDL document with the description of the hire method.

```
<tuple link="http://www.mmds.org/hire" type="WDSL" ctx="parent"
TS1="10" TC="15" TS2="20" TS3="30">
<content>
<message name="hirecrew">
   <part name="filmcrew" type="xs:string"/>
</message>

<portType name="hirefilmcrewservices">
  <operation name="hirefilmcrew">
      <input message="hirecrew"/>
  </operation>
</portType>

<binding type="hirefilmcrewservices" name="hs1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://www.mmds.org/hire"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
</content>
<metadata> <owner name="http://www.mmds.org"/> </metadata>
</tuple>
```

5.4 WSIL

The Web services Inspection Language [13], is complementary to the registry approaches considered so far. WSIL is a distributed metadata model for web service information. It assumes no restrictions of the published content. The WSIL provides a method for aggregating different types of web service descriptions in a single document. WSIL serves two purposes: First, it defines an XML format for listing references to existing service descriptions. Second, it defines a set of conventions so that it possible to locate WS-Inspection documents.

Each web service provides a WSIL file at a specified location. WSIL can be regarded as business cards containing arbitrary information, for example, HTTP links to ontology documents, WSDL documents, etc. The example below illustrates a WSIL document containing links to the hire API of our case study and UDDI identifier for detailed information about the hire services.

```
   <inspection
targetNamespace="http://schemas.xmlsoap.org/ws/2001/10/inspection/"

xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/
"
      xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
     <link referencedNamespace="urn:uddi-org:api">
       <wsiluddi:businessDescription location=
       "http://www.mmds.org/hire ">
         <wsiluddi:businessKey>3BF0ACC0-BC28-11D5-A432-0004AC49CC1E<
       /wsiluddi:businessKey>
         <wsiluddi:discoveryURL useType="businessEntity">
           http://www.mmds.org/uddi?businessKey=
       3BF0ACC0-BC28-11D5-A432-0004AC49CC1E
         </wsiluddi:discoveryURL>
       </wsiluddi:businessDescription>
     </link>
     <service>
       <name>MMDS Inc</name>
       <description referencedNamespace="urn:uddi-org:api">
         <wsiluddi:serviceDescription location=
       "http://www.mmds.org/hire">
           <wsiluddi:serviceKey>52946BB0-BC28-11D5-A432-0004AC49CC1E<
       /wsiluddi:serviceKey>
           <wsiluddi:discoveryURL useType="businessEntity">
             http://www.mmds.org/uddi?businessKey=
       3BF0ACC0-BC28-11D5-A432-0004AC49CC1E
           </wsiluddi:discoveryURL>
         </wsiluddi:serviceDescription>
       </description>
     </service>
   </inspection>
```

5.5 Human View on Web services Data Models

From a human point of view, all presented Web services data models share large similarities regarding the human requirements, respectively the human view. Each presented data model is human readable and allows meaningful (from a human point of view) annotations or comments to the data structures. These descriptions allow humans to gain a better understanding of the underlying data structure. One thing missing in all approaches is a “direct” semantic approach. None of the approaches provides an implicit meaning or context information. Take for example the WSDL description of the hire method. The document is human readable but it does not state the context where the method is going to be used. Additionally, it only provides a syntax definition but no semantic description of the meaning of the method. It relies on “meaningful” names of ports and methods. The same description is meaningless for a human when using arbitrary combinations as method identifier and port names although the service description falls into the same category.

6 Publishing of Web services

The publishing of Web services is closely related to the underlying data structure. Each data structure (UDDI, ebXMl, and WSDA) defines what kind of data is being published. This section illustrates the different approaches for the publishing of Web services taken by UDDI, ebXMl, and WSDA from a technical point of view.

6.1 Publishing of Web services in UDDI

The UDDI publisher API provides the interface for the publishing and management of web services. It provides functions for the addition of new web services as well as functions for the adaptation of existing web services using save_XX API calls. The example below shows how to register our case study by creating a new businessEntity with name, description, and contact information:

```
   <save_business>
   <businessEntity businessKey="">
     <name> Martins Movie Managing services </name>
     <description xml:lang="en">
       The triple M S offer a variety of Web services for the
 management of movies. The services include the selection of different
 film crews, like stuntmen, makeup artists and so on.
     </description>
     <contacts>
       <contact useType="US general">
         <personName>Martin Marty</personName>
         <phone>1 800 CALL MMMS</phone>
         <email useType="">office@mmms.org</email>
         <address>
           <addressLine>MMMS</addressLine>
           <addressLine>1000 Bollywood Avenue</addressLine>
           <addressLine>Bombay 1000</addressLine>
         </address>
       </contact>
     </contacts>
   </businessEntity>
   <save_business>
```

6.2 Publishing of Web services in ebXML

To register Web services, a Web services provider must contact the Lifecycle Manager and provide the information about the Web services to register. The registration of Web services [20] does not affect every part of the ebXML registry data structure. To register a Web services, it is necessary to prevent data for the following classes in the ebXML registry:

- services
- serviceBinding
- SpecificationLink

The publishing of our case study from section 2 in an ebXML registry can be accomplished as shown in the example below. The actual Web services description holds an external WSDL document, which can be retrieved using the specified access URI that it is referenced by.

```
   <SubmitObjectsRequest>
     <LeafregistryObjectList>
   <service id="MMDS Inc">
       <Name>
     <LocalizedString lang="en_US" value = "Martins Movie Management
services"/>
       </Name>
     <Description>
        <LocalizedString lang="en_US" value = "This Web service will
accept purchase orders for MMMS Corporation.  It will validate the
contents of each purchase order, and, if valid, will process the
purchase order and automatically generate an Invoice."/>
     </Description>
     <serviceBinding
accessURI="http://www.mmms.org/getservicesDescription">
         <SpecificationLink
specificationObject="wsdlForgetservicesDescription ">
       <UsageDescription>
         <LocalizedString lang="en_US" value = "This is the WSDL
document that describes the getservicesDescription Web services"/>
       </UsageDescription>
     </SpecificationLink>
     </serviceBinding>
      </service>
      <ExtrinsicObject id="wsdlForPurchaseOrder" mimeType="text/xml">
    <Name>
       <LocalizedString lang="en_US" value = "The WSDL document for
the MMMS getservicesDescription web service"/>
      </Name>
        </ExtrinsicObject>
     </LeafregistryObjectList>
   </SubmitObjectsRequest>
```

6.3 Publishing of Web services in WSDA

The publishing of Web services in WSDA is executed by the consumer interface. This interface provides a publish method using a tupleset as input. A tupleset contains several tuples, each tuple identified by a unique key that consist of the pair content link and context. The example below shows a tupleset example for the publishing of the hire service of the director. Note that the example makes use of SWDSL, a simplified variant of WSDL, proposed in [8]:

```
   <tupleset>
   <tuple link="http:// www.mmds.org/hire" type="service" ctx="parent"
   TS1="10" TC="15" TS2="20" TS3="30">
   <content>
   <service>
   <interface type="http://www.mmds.org/Presenter-1.0">
```

```
<operation>
<name>XML hirefilmcrew(filmcrew filmcrew)</name>
<bind:http verb="GET" URL="https://www.mmds.org/hire"/>
</operation>
</interface>
<interface type = "http://www.mmds.org/XQuery-1.0">
<operation>
<name> XML query(XQuery query)</name>
<bind:http URL="http://www.mmds.org/"/>
</operation>
</interface>
</service>
</content>
<metadata> <owner name="http://www.mmds.org"/> </metadata>
</tupleset>
```

6.4 Human View on Web services Publishing

The human view on Web services publishing depends on the abstraction level taken on Web services. On a programming level technical details are of importance, leading to a concrete rather than abstract view on Web services publishing. Method descriptions, often on a syntactical level, are the main issue. Set directly on top of the programming level, Web services can be considered as components with several methods. Components are a more abstract concept and are used for example by system designers. On a management level, the Web services themselves are of interest. Structural details, like for example the component structure of a Web services or syntactical details of methods, are not important. The main issue is the capability of a Web services. The management level focuses on abstract descriptions of Web services.

7 Discovery of Web services

The following section shows the discovery of Web services using the provided discovery mechanisms of UDDI, ebXML, and WSDA. At the end of the section, a summary compares the human requirements for Web services discovery and the approaches taken by UDDI, ebXML, and WSDA.

7.1 Discovery of Web services in UDDI

The UDDI inquiry API specifies the functions for the discovery of web services. The find_XX API calls provide an overview of registration data based on a variety of search criteria. The easiest way to query a UDDI registry is by keyword based search for a business name. The example shows how to search a UDDI registry for our case study using the name as the search criteria:

```
<uddi:find_business generic="2.0" maxRows="10">
  <uddi:name>MMMS Inc</uddi:name>
```

```
   </uddi:find_business>
```

Other ways to query a UDDI registry are the use of categorization elements (categoryBag), or the use of identification elements (indentifierBag):

```
   <find_business xmlns = "urn:uddi-org:api_v3"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
   <findQualifiers>
   <findQualifier>
   uddi:uddi.org:findQualifier:approximateMatch
   </findQualifier>
   </findQualifiers>
   <identifierBag>
   <keyedReference
   keyValue = "%"
   tModelKey = "uddi:ubr.uddi.org:identifier:dnb.com:D-U-N-S"/>
   </identifierBag>
   </find_business>
```

The hierarchical data model of the UDDI registry is reflected in the way information is obtained from a UDDI registry. Because each entity in the UDDI data model is identified by an UUID, it is possible to search for registry entries by use of their UUID. If the key of a registered service is known ahead, then its is possible to obtain information directly by the use of the get_XX API calls. The example searches for detailed information about a business with the help of the UUID:

```
   <uddi:get_serviceDetail generic="2.0">
     <uddi:serviceKey>860eca90-c16d-11d5-85ad-
801eef208714</uddi:serviceKey>
   </uddi:get_serviceDetail>
```

7.2 Discovery of Web services in ebXML

The ebXML QueryManager interface provides all necessary methods for the interaction with an ebXML registry. The QueryManager implements two kinds of query mechanisms:

- Filter Query
- SQL Query

The Filter Query mechanism supports a XML based query syntax that specifies a set of filter classes. Each filter specifies requirements for the successful matching of registry entries. The Filter Query interface supports a XML syntax that allows the definition of a set of class filters. The set of class filters is matched against the registry entries. The example above shows a Filer Query example where all registry entries from film businesses in Austria are retrieved:

```
   <registryEntryQuery>
   <registryEntryFilter>
   status EQUAL "Approved"
   </registryEntryFilter>
   <HasClassificationBranch>
   <ClassificationNodeFilter>
   id STARTSWITH "urn:ebxml:cs:industry" AND
   path EQUAL "Industry/filmbusiness"
   </ClassificationNodeFilter>
    <ClassificationNodeFilter>id STARTSWITH "urn:ebxml:cs:geography"
AND path EQUAL "Geography/Europe/Austria"
   </ClassificationNodeFilter>
   </HasClassificationBranch>
   </registryEntryQuery>
```

The SQL Query interface supports a basic subset of the SQL SELECT statement as described by the SQL-92 standard [30]. Following a relational data model, instances of classes from the ebXML data model can be mapped to tables with columns according to their attributes. The following example shows a simple SELECT statement, where all instances of the registryObject (respectively their IDs) having the string "MMDS" in their name and the string "movie director" in the description, are being retrieved:

```
SELECT r.id from registryObject r, Name n, Description d where n.value
LIKE '%MMDS%' AND d.value LIKE '%movie director%' AND r.id = n.parent
AND r.id = d.parent
```

7.3 Discovery of Web services in WSDA

The discovery of Web services is carried out by two interfaces, the MinQuery and the XQuery interface. The MinQuery interface allows basic query support, where the complete tupleset is returned (see example). The XQuery interface provides XQuery support. XQueries allow complex - non trivial - searching for services. When a peer receives a XQuery request the XQuery expression is evaluated against the content portion of the data tuple. The following example shows a query that finds all available hire services provided by directors:

```
LET $s:=/tupleset/tuple[@type="service"]
         /content/service/interface[@type="http://www.mmds.org/hire"]
FOR $director IN /tupleset/tuple[content/service/interface/@type=$s
RETURN $director
```

7.4 Human view on Web services discovery

From a human point of view, the presented discovery mechanisms differ in their expressive power, their complexity, and their usability. The UDDI registry uses key lookups with simple qualifiers. Searches in UDDI can be carried out in several ways. The simplest possibility is a keyword based search. Advanced queries can be expressed by the use of categorization info. Categorization information can be referenced with tModels, respectively by their UUIDs. A taxonomy based search in a UDDI registry needs the tModel UUID of the categorization.
ebXML allows filter based search queries. Filters are a very powerful method to search in the ebXML registry. Filters support hierarchical searches and allow users to

combine different filter for a hierarchical query. Another possibility is the use of SQL based search queries. A subset of SQL select statements enables user to construct queries over several classes respectively tables at once. WSDA offers two interfaces for the search. The MinQuery interface allows for basic search queries in the way of select-all expressions. The XQuery interface implements the XQuery language that allows user to search with the help of complex XQuery expressions. XQuery expressions are a very powerful method for querying a registry. UDDI, ebXMl, and WSDA assume at least basic knowledge about the underlying data structure when searching for registered items. The more a user knows and understands the data model, the better the search results are. The main problem is a complex data model. The ebXML data model is rather complex and needs a good understanding of the structure when performing advanced searches.

7.5 Summary

The four different approaches share the same goal: to support the publishing and discovery of Web services. The actual implementations - from a central registry with strong limitations regarding the published content to a fully distributed registry without any limitations regarding the published content - differ in many ways.

The UDDI and ebXML registry implementations follow a centralized approach where a central entity manages the registry information. In future, both registry architectures will implement a distributed architecture as well to overcome scalability problems. In contrast to UDDI and ebXML, WSDA offers a flexible way to implement registries - a dedicated peer can store arbitrary registry information or each peer can publish its own Web services registry service. Complementary to these approaches, the WSIL offers a simple distributed metadata service, where each service peer publishes a WSIL document on its own at a "well known" address.

The UDDI approach implements a hierarchical data model. The data model follows a top down approach, a businessEntity element stores basic business information along with several nested sub elements for detailed technical information about the business. registry elements can also point to external resources using tModel elements and thus accomplishing extensibility. In UDDI, the categoryBag and the indentifierBag elements allow a simple classification of registry entries, without the possibility of hierarchal taxonomies.

The ebXML registry implements a data structure with classifications and a hierarchical classification schema. The ebXML registry assumes no limitations about the content being stored. The slot element allows to extend the data model using key name pairs. registry entries can also point to external resources, for example a link to a WSDL document can be stored in an ebXML registry using the externalLink element.

Both data models share similarities, for example, when registering a basic Web services. The main difference between these two data models lies in the way how registered objects are categorized. ebXML offers a built-in extensible category schema, while UDDI relies on tModel links to an external classification schema. The ebXML registry supports collaboration and coordination protocols (CPP, CPA), UDDI does not offer similar capabilities.

The WSDA implements a tuple based data model. Each tuple contains several attributes and can contain arbitrary content. Actual registry information is stored in the content part of a tuple, which itself may point to an external registry. WSDA does not support classification or semantic data directly. Due to the arbitrary data model it is possible to make use of XQuery expressions that match certain hierarchical criteria, when it is applied to XML based tuple content.

Compared with ebXML and UDDI, WSDA takes a complementary position. WSDA does not provide a data model for the actual registry entries; it enables Web services provider only to publish arbitrary descriptions into the tuple space. Data tuples can be highly dynamic, their lifespan is determined by several timestamps, whereas the lifespan in UDDI and ebXML is not limited, because once a Web service is registered, it stays in the registry as long as the registry is inline or the Web services provider deletes the Web services registry entry.

The WSIL acts as container for arbitrary web service descriptions or registry entries. This approach is similar to the approach taken by WSDA, where registry information can be stored in the content part of a tuple.

The UDDI inquiry API provides several API functions for searching the registry. The API allows to search in several ways for a business, for example, using keyword based name queries, or looking for a business using external identifiers.

The search capabilities of ebXML registries are more powerful. The query interface allows complex search queries with the help of filter expressions or basic SQL select statements.

The Web services Discovery Architecture follows a distributed approach where every peer is capable to offer its own service description. registry capabilities can also be overtaken by a dedicated peer. WSDA offers two query interfaces the XQuery interface that provides XQuery support and the MinQuery interface that provides basic selection support for the discovery of web services.

WSIL does not support any direct query interfaces. It is mainly a metadata container of distributed nature and specifies no discovery interfaces.

None of the approaches implements a built in QoS schema. QoS attributes can be referenced by UDDI, ebXML, and WSIL by external QoS descriptions. WSDA allows to store arbitrary QoS descriptions as tuples, but does not offer direct QoS support. Table 3 compares the aforementioned approaches:

	UDDI	**ebXML**	**WSDA**	**WSIL**
Registry Architecture	Centralized/ Decentralized	Centralized/ Decentralized	Centralized/ Decentralized	n.a.
Fault tolerance	Low	Caching	Caching	n.a.
Scalability	Medium	n.a.	High	n.a.
Complexity	Simple	High	Simple	Simple
Platform	Various	Various	Various	n.a.
Search capabilities	Poor	Good	Rich	n.a.
Service description	No, external	Yes	No, external	Yes
Semantic data/	Possible	Yes	Possible	Possible

Ontologies				
Extensibility	Low	High	High	High
Test registries	Yes	Yes	No	n.a.
Overhead	High	High	Medium	Low
Replication of registries	Yes	Yes	No	n.a.
QoS	No	No	No	No
Dynamic registry Entries	No	No	Yes	n.a.

Table 3. Comparison of registry features

8 Conclusion

From a human view the main problem of current Web services registry technologies is the lack of human interpretable information. Despite of providing human readable information in form of tag based documents, the presented information remains rather formal and abstract. The enrichment with informal inline descriptions and categorizations eases the understanding but is not sufficient. A possible solution is an active data model, which shows potential usage of the described Web services based on examples or working scenarios.

From a machine view, semantic markup languages such as DAML+OIL enrich registries and provide meaningful information. Ontologies for arbitrary content like DAML-S [15] provide UDDI registry entries with semantic information. Technically DAML-S profiles are mapped into UDDI registries with the help of tModels. A DAML-S matching engine uses ontology based information for search requests to obtain UDDI keys which are in turn used to retrieve the service descriptions from UDDI registries.

Another example is EDUTELLA [17] that is built on top of the JXTA P2P framework and provides a peer to peer system with service descriptions in RDF. EDUTELLA allows complex query operations based on RDL-QEL, provided on several levels of complexity.

The METEOR-S system implements specialized ontologies, called registry ontology. Registry ontologies capture properties of registries. The ontology data is useful for the discovery of registered services. It is possible to update registry ontologies with another registries ontology to obtain a combined ontology thus implementing relationships between services of different registries.

Another approach is taken in [31]. Directory information is enriched by context aware data which is represented by a Multidimensional OEM graph [32]. This data structure allows to model different facets under different contexts thus providing a hierarchical structure.

Acknowledgements

This research was supported in part by a research award for Development Methods for Dynamic Web services Workflows of the Chamber of Commerce Vienna (Wirtschaftskammer Wien).

9 References

[1] Universal Description, Discovery and Integration: UDDI Technical White paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf. 2000
[2] W3C. Web services Architecture W3C Working Draft 8 August 2003. http://www.w3.org/TR/2003/WD-ws-arch-20030808/wsa.pdf. 2003
[3] 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
[4] 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. 2002
[5] Fabio Casati, Ming-Chien Shan. Dynamic and adaptive composition of e-services. Software Technology Lab, Hewlett-Packard Laboratories. 2001
[6] UDDI Version 2.03 Data Structure Reference. http://uddi.org/pubs/DataStructure_v2.htm. 2002
[7] UDDI Version 3.0.1 http://uddi.org/pubs/uddi_v3.htm. 2003
[8] Wolfgang Hoschek. Peer-to-Peer Grid Databases for web service Discovery. CERN IT Division. 2002
[9] OASIS/ebXML registry services Specification v2.5. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf. 2003
[10] OASIS/ebXML Technical Architecture Specification. http://www.ebxml.org/specs/ebTA.pdf. 2001
[11] OASIS/ebXML registry Information Model v2.0 http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf . 2001
[12] 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
[13] Keith Ballinger, Peter Brittenham, Ashok Malhotra, William A. Nagy, Stefan Pharies. Web services Inspection Language (WS-Inspection) 1.0. IBM, Microsoft. 2001
[14] Mike P. Papazoglou, Bernd J. Krämer, Jian Yang. Leveraging Web-services and Peer-to-Peer Networks. INFOLAB - Tilburg University, FernUniversität Hagen. 2003
[15] DAML-S Coalition. DAML-S: Web services Description for the Semantic Web. 2002
[16] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara. Importing the Semantic Web in UDDI. Carnegie Mellon University. 2002
[17] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambrjörn Naeve, Mikael Nilsson, Matthias Palmer, Tore Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. 2001
[18] W3C. SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. 2003
[19] W3C. WSDL, Web services Description Language. http://www.w3.org/TR/2002/WD-wsdl12-20020709/. 2002
[20] ebXML registering a Web services within a ebXML registry.

http://www.oasis-open.org/committees/download.php/1636/OASIS-registry%20TC%20-%20Registering%20Web%20services%20in%20an%20ebXML%20registry.doc. 2003

[21] Hypertext Transfer Protocol - HTTP/1.1. IETF RFC 2616. UC Irvine, Digital Equipment Corporation, MIT. 1999

[22] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup services for Internet Applications. University of California, Berkeley, MIT Laboratory for Computer Science. 2002

[23] Using WSDL in a UDDI registry, Version 2.0 http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v200-20030627.htm. 2003

[24] Christina Schmidt, Manish Parashar. A Peer-to-Peer Approach to Web services Discovery. Department of Electrical and Computer Engineering, Rutgers University. 2003

[25] W3C. XML Extensible Markup Language. http://www.w3c.org/XML. 2000

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

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

[30] W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/2003/WD-xquery-20031112/. 2003

[31] Christos Doulkeridis, Efstratios Valavanis, and Michalis Vazirgiannis. Towards a Context-Aware services Directory. Database Systems Laboratory Department of Informatics Athens University of Economics and Business (AUEB) 10434 Athens. 2003

[32] Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-dependent Information on the Web. In Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering (CAISE'02), Toronto, Canada. 2002.