Distributed and Parallel Databases, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s10619-005-2460-y

A View Based Analysis on Web Service Registries

SCHAHRAM DUSTDAR dustdar@infosys.tuwien.ac.at
MARTIN TREIBER e9426464@student.tuwien.ac.at
Distributed Systems Group, Vienna University of Technology

Recommended by: Dimitrios Georgakopoulos
Published online: 12 July 2005

Abstract. Web services registries are a cornerstone for the emerging service-oriented architecture and constitute a critical resource for Web services. 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 Web service based views. We use these views to show the different requirements and to illustrate the different abstractions when comparing Web service registries. The human view on Web service registry architectures is illustrated with the help of a case study. The Web service view on Web services registry architectures is illustrated from a software-service point of view. The data model of Web service registries is described in detail from a machine based view. The corresponding human view is described from an abstract level. Web service publishing and discovery are compared from a human and a Web service based view. Finally, we present a working example that uses our methodology to compare different Web service registries and to explain the different views introduced in this paper.

Keywords: Web service registries, service-oriented architecture

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 [21]. According to the W3C, Web services [31] 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 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 service brokers, respectively Web service 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 service broker respectively a Web service registry. The service requestor wants to fulfill a certain task with the help of one or more Web service(s). In order to locate Web services, the Web service requestor contacts a service broker to search for Web services. When an adequate Web service is found the Web service requestor uses the information of the Web

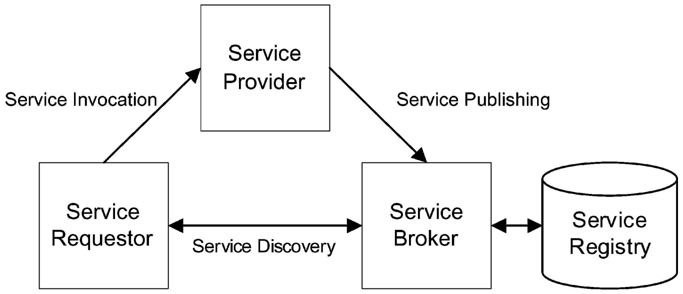

Figure 1. Conceptual overview of Web services.

service broker to invoke the Web services. The service broker (registry) stores information describing Web services provided by Web service providers in a registry (repository). The Web service registry allows users to search for Web services and to publish Web services descriptions.

The selection of adequate Web service registries is important at the design time of Web service composed systems. We consider Web service registries as critical components during the design time of a Web service oriented software system. Since different Web service registry implementations exist, the selection of an adequate Web service registry is important. Comparing different Web service registries two main questions emerge:

– What are the typical requirements for Web service registries?
– Which criteria are reasonable for Web service registry selection?

Our work focuses on requirements of Web service registries. In particular, we address the different perspectives on Web services and the associated context dependent requirements. We introduce the notion of a *view* that serves as a means for the comparison of Web service registries. A view structures requirements of Web service registries in arbitrary dimensions. It serves as a method to capture the requirements that lead to certain Web service registry architectures, respectively, Web service data models. This paper compares different approaches of Web service registries from the human and the Web services perspective. These perspectives (views) provide two dimensions: Architecture and Data Model. The architecture describes the conceptual structure of a Web service registry. The data model describes the type of data and the data structure implemented by a Web service registry. A view, furthermore, enables the creation of differing perspectives on Web service requirements.

The reminder of the paper is organized as follows: Section 2 presents requirements of Web service registries and introduces the two views on Web service registries and discusses different Web service architectures and data models accordingly. Section 3 discusses Web services architectural models. Section 4 analyses Web service registry data models. Section 5 provides a case study from the movie business which is used to illustrate our framework. We compare and discuss the different approaches of Web services registries regarding their data models.

2. A framework for Web Service registry comparison

The requirements Web service registries face are similar to the requirements of the Web services architectures [31]. The seven top level requirements described in [32] serve as starting base for our analysis of requirements concerning Web service registries. We refine these requirements with regard to Web service registries. We consider additional requirements, for example, the expressiveness of the Web service description as important and include them into our considerations. Table 1 presents a summary of these refined, respectively extended, requirements.

Table 1. Web service registry requirements.

Requirement	Description
Interoperability	Defines the ability of exchanging and using information between different heterogeneous Web service registry environments
Reliability	Defines the degree to which a Web service registry is capable maintaining the service at a given service quality
Integration with the World Wide Web	Defines the degree to which a Web service registry is consistent with the current and future evolution of the World Wide Web
Security	Defines the level of security necessary to access a Web service registry
Fault tolerance	Defines the ability of a Web service registry to continue normal operation despite the presence of hardware or software faults
Scalability	Defines how well a Web service registry responses to increasing load
Availability	Defines the probability of successful Web service registry invocations
Expressiveness of query language	Defines the level of quality regarding the expressiveness of the used query language
Expressiveness of Web service description	Defines the level of quality regarding the used Web service description language
Extensibility of data model	Defines the degree of possible data model extensions regarding Web service descriptions
Transient Web services	Defines the ability of Web service registries to administer Web services that provide limited access regarding online time
Management and provisioning	Defines the ability of Web service registries to provide for a manageable, accountable environment for Web service registry operations

Selecting reasonable criteria for the rating of Web service registries depends on the actual view on the Web service registry. Not every requirement is a reasonable selection criterion for a given view on Web service registries. We propose a comparison framework that unifies the aforementioned aspects of Web service registry requirements using an abstraction that we call view. This abstraction allows the comparison of different Web service registries regarding different requirements along several dimensions. We illustrate the use of the view based Web service registry comparison with respect to two dimensions. We investigate (1) the Web service registry architecture and (2) the Web service registry data model with the two typical functions of Web service registries, i.e., the Web service discovery and the Web service publishing.

We consider the requirements under two different views, the human and the Web service based view. We refine the requirements with regard to the actual view, because every view provides own requirements and different abstractions. Views may include differing semantics of the same abstractions. Consider, for example, the notion of security. From a Web service view security involves the use of certain encryption algorithms or the use of encrypted communication channels. From a human view the same concept may apply to securing the Web service publishing by the use of passwords.

We start our discussion with the human view on Web service architectures. The human view on Web service registries has requirements without providing detailed information concerning technical issues. Requirements, for example, such as extensibility of data model, support for transient Web services, the used standards regarding Web service description encoding or the Integration with the World Wide Web are technical issues.

In contrast, the Web service view on Web service registry architectures focuses on technical issues regarding the requirements. Although the Web service view shares some of the requirements of the human view, the Web service view provide different semantics regarding the actual requirements. The Web service view relies on attributes that allow numeric quantification. Requirements, like reliability, performance and availability provide concrete values that define the performance. Other requirements like security or supported standards include, for example, concrete security protocols or encoding schemata that are supported. We summarize these requirements in Table 2.

The second dimension is provided by the view on Web service registry data models. We start with the human view on Web service registry data models. The human view on Web service registry data models is limited by human readable, respectively human understandable, information. The corresponding requirement, i.e., the expressiveness of Web service descriptions, defines the degree to which Web service descriptions provide human interpretable information. Human interpretable information is usually unstructured text 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. Human understandable data can further be structured, for instance, into data that provides technical description, such as references to standards, etc.

Web services description may also provide related information like introductive texts about the Web services in a business context. 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

Table 2. Human and Web service view on Web service registry requirements.

Requirements	Human view	Web service view
Interoperability	n.a	Defines the ability of exchanging and using information between different heterogeneous Web service registry environments
Reliability	Defines the percentage of successful service invocations regarding the Web service registry basic functionality	Defines the degree to which a Web service is capable maintaining the service at a given service quality
Integration with the World Wide Web	n.a	Defines the degree to which a Web service registry is consistent with the current and future evolution of the World Wide Web
Security	Defines security mechanisms for the access of Web service registries that prevent security attacks of the Web service registry	Defines the level of security necessary to access Web services
Expressiveness of Query language	Limits the way how humans can search for Web services	Defines the level of quality regarding the used query language
Expressiveness of Web service description	Limits the degree to which humans can understand Web service descriptions	Defines the level of quality regarding the used Web service description language
Fault tolerance	Defines the ability of Web service registries to continue normal operation despite the presence of hardware or software faults	Defines the ability of a Web service registry to continue normal operation despite the presence of hardware or software faults
Scalability	Defines how well Web service registries response to increasing number of Web service registry entries and to increasing number of search queries	Defines how well a Web service registry responses to increasing load
Extensibility of data model	n.a	Defines the degree of possible data model extensions regarding Web service descriptions
Transient Web services	n.a	Defines the ability of Web service registries to administer Web services that provide limited access regarding online time
Availability	Defines the percentage of time that Web service registries are online and operational	Defines the probability of successful Web service invocations
Management and provisioning	Defines the ability of Web service registries to provide for a manageable, accountable administration interface for Web services registries operations	n.a

Table 3. Human and Web service view on registry data model requirements.

Attribute	Human view	Web service view
Expressiveness of Query Language	Limits the way how humans can search for Web services	Defines the used query language(s) to support the Web service discovery
Expressiveness of Web service description	Limits the degree to which humans can understand Web service descriptions	Defines the level of quality regarding the used Web service description language
Extensibility of data model	n.a	Defines the degree of possible data model extensions regarding Web service descriptions
Semantic Meta Data	n.a	Defines the used language(s) for the support regarding semantic information

about 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".

Functional aspects of Web service registries consider the expressiveness of the query language. The way how human user can formulate the queries influences the success of Web service discovery. From a human view, it is necessary to provide query languages that support the expression of search queries as closely as possible to human requirements.

The Web service view on Web service data models focuses on technical issues. In general, the Web service view depends on well structured data. In contrast to the human view on Web services registry data models, the Web service view addresses the actual used data model (hierarchical, object oriented, relational, etc.) and how this data model represents the Web service descriptions. The used data model defines the level of expressiveness regarding the query languages the degree to which it is possible to extend the existing data model. A related issue is semantic information. Consider the example of a user who wants to find a Web service that sells airline tickets between two given cities and accepts a particular credit card. Currently, this task is usually performed by a human who searches for a corresponding Web service. He/she has then to determine if the found Web service satisfies the constraints. With semantic markup of Web services, the information necessary for Web service discovery could be specified as computer-interpretable semantic markup and a semantic-enhanced Web service registry can be used to locate the Web service automatically. We summarize the human and the Web service view in Table 3.

3. Web services registry architectures

This section discusses the architectural style of Web service registries. The architectural style of a service-oriented system using Web services defines how a Web service registry is actually implemented. The implementation of Web services influences the message interaction schema between Web service registry, Web service provider, and Web service requestor. Generally, Web services registries can be classified with regard to their architecture: (a) Centralized, (b) Decentralized, or (c) Hybrid. Each of these different architectural styles provides certain strengths and weaknesses regarding scalability, fault-tolerance, administrative overhead, complexity, and performance. This section gives an overview of the different Web service architectures and provides examples for every type of architecture. We conclude this section with a view based architectural comparison of the different Web service registry approaches. The view concept emphasizes the differences between the human and Web service perspective of Web service registry architectures.

3.1. 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 service 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 best know example for a centralized Web service is UDDI (Universal Description, Discovery and Integration). UDDI 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 [27], HTTP [9], TCP/IP) and is set on top of an interoperating stack. UDDI focuses mainly on the discovery of services using a centralized Web service registry. Web service descriptions are not part of the UDDI specification. Service descriptions such as WSDL can be referenced by UDDI registry entries using tModels [25].

The SELF-SERV project [2, 3] is an example for using a centralized UDDI [22–24] based registry (Figure 2). The Services Manager component consists of three modules, namely the Services Discovery Engine [17], 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 [28] and SOAP [29] 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.

Another example for a centralized Web service registry architecture is the ebXML (electronic business XML) standard [12]. It 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. 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

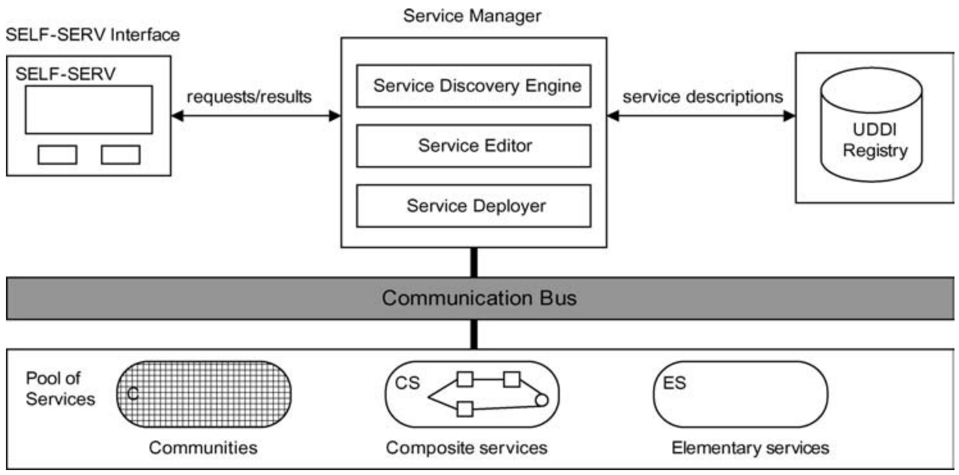

Figure 2. SELF-SERV and UDDI.

in scope. An ebXML registry is capable of storing arbitrary data, for example, Business Process Models [4], CPP [7] or CPA [34].

The ebXML registry offers two separate interfaces, the LifeCycleManager Interface and the QueryManager Interface. 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.

3.2. 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 as the Web service provider leaves the network, the registry entry is not valid anymore, since the Web services registry entry is not available. This implies a dynamic registry structure where the lifespan of a registry entry is limited by the connection time to a peer network.

Schmidt and Parashar [16] present 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 [18] 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 a 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 node sends a join message which is routed across the network and is then inserted into the network structure.

The Web services Discovery Architecture [20, 33] 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 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. The XQuery interface provides XQuery support. 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, namely the Pull registry, the Push registry and 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.

3.3. Hybrid architecture

The federated approach distributes Web services registration information among different entities in a peer to peer fashion. Dedicated nodes of the network, i.e., 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 like in a centralized Web services environment since they are not aware of the distributed nature of the Web services registry. 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 service. A Web services provider can publish a Web service 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.

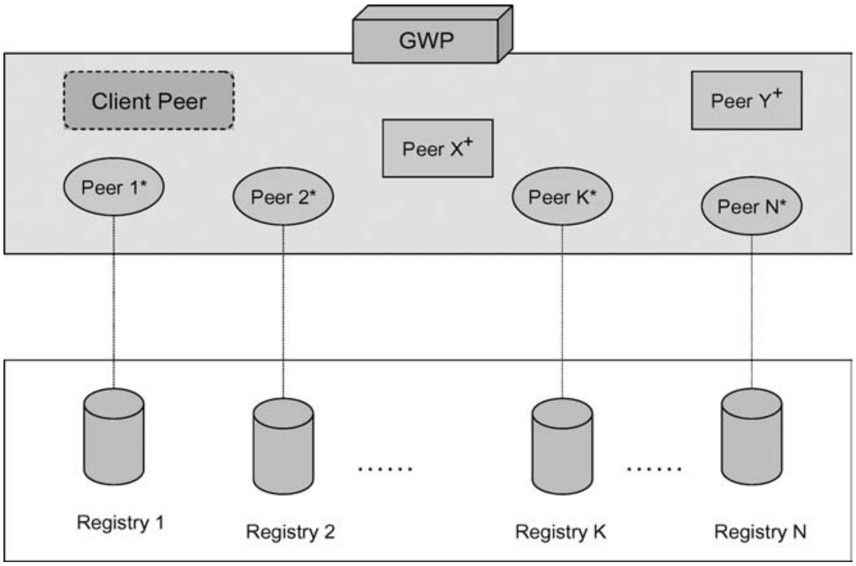

Figure 3. METEOR communication layer overview.

The METEOR-S [26] project implements a distributed registry structure as depicted in Figure 3. METEOR support four different types of peers: gateway, operator, auxiliary, and client peers. Each operator peer controls a local registry and provides operator services. The operator peer provides advanced Web services discovery mechanisms based on ontological information. 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.

Because of the gateway peer, METEOR can be classified as a 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 METEOR features are available, for example, it is not possible for new registries to join the network.

Papazoglou et al. [15] introduce 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 4).

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

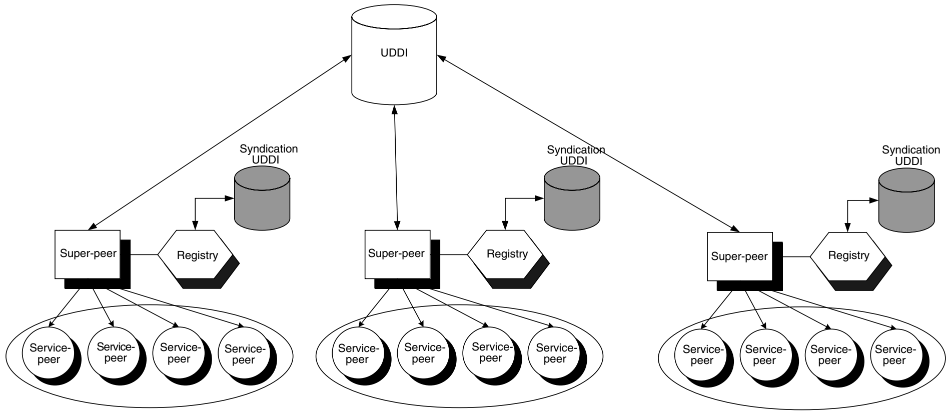

Figure 4. Web services Syndication overview.

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.

3.4. View based comparison of the architectural styles

The human and Web service view on Web service architectures provide the same results regarding common requirements. Although they have different semantics, the impact on Web service architectures is almost the same.

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 and provides limited scalability. 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.

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 peer, since the amount of data is less, compared to a

Table 4. Human and Web service view based comparison of Web service registry architectures. Attributes written in italics are available either in the human or the Web service view.

Attributes	Centralized	Hybrid	Decentralized
Scalability	Low	High	High
Fault-tolerance	None	Yes	Yes
Reliability	Yes	Yes	No
Performance	Low	High	Medium
Availability	Medium	High	High
Security	Good	Good	Fair
*Integration with the World Wide Web**	Good	Good	Fair
*Management and Provisioning**	Very Good	Good	Fair
*Support of transient Web services**	Poor	Poor	Good
*Interoperability**	Good	Fair	Fair

*Web service view only.

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. The failure of a peer does not affect any other peer, because every 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, since 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 service 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. 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 Web services since network bandwidth is consumed by the search messages. Table 4 presents detailed results of the comparison.

4. Web service registry data models

Another relevant difference between Web service registries concerns the type of information stored in a Web service 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.

We discuss the differences between Web service registries concerning their data models. Web service registries implement different data models to store registry information. Persisted data ranges from simple informal Web services descriptions to formal ontological structured information. The following section presents a short overview on UDDI, ebXML, WSDA, and WSIL concerning their data models and compares the different approaches.

4.1. UDDI data model

The UDDI data model [22] is a hierarchically-structured data model. It provides a "top-down" approach, where information about a Web service is divided into several categories with each category offering more detailed information about the registered Web services. Each entity in the data model is identified by a unique universal identifier (UUID). 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 data model consists of businessEntity structures that encapsulate information about Web services. A businessEntity holds businessService structures that describe 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 a URL) and references to tModels.

4.2. ebXML data model

The ebXML data model provides a class hierarchy that allows the modeling of registry entries. Since the data model also includes other aspects of electronic business, the data ebXML model is much broader in scope than for example UDDI. Generally speaking, the ebXML registry data entries provide metadata about registry objects. These entries are not limited to Web service descriptions. ebXML is capable of modeling arbitrary data, like for example UML diagrams, etc. The data model itself is organized into 17 classes that enable for example the creation of taxonomies of registry entries or the grouping of related registry objects.

4.3. 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 that consists of five different fields. 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. 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. 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. Every registry entry points to the external description of the Web service.

4.4. WSIL

The Web services Inspection Language is complementary to the registry approaches considered so far. WSIL provides a distributed metadata model for web service information. It assumes no restrictions of the published content. 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 is 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.

4.5. View based comparison of Web service data models

We start our discussion of the different Web service data models with the human view. The hierarchical UDDI data model supports the human need for general information about Web services. It provides basic information about the Web service provider and enables the stepwise refinement of search criteria by following the data model. The main drawback is the limited search capability. The UDDI registry data model allows to search in several ways for Web services, for example, using keyword based name queries, or looking for a Web service using external identifiers. The use of external identifier is not very comfortable and includes the handling of UUIDs that reference the actual information. UDDI supports only the keyword based search and the browsing of predefined categories. The publishing of Web services is limited to technical aspects; concrete functional considerations are out of the scope of UDDI registries.

From a human view, the ebXML registry offers a more flexible data structure with classifications and a hierarchical classification schema. The ebXML registry data model assumes no limitations about the content being stored. The slot element allows the extension of 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. Compared with UDDI, the search capabilities of ebXML registries are more powerful. ebXML supports complex search queries with the help of filter expressions or basic SQL select statements.

Both data models share similarities, for example, when registering 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), whereas UDDI does not offer similar capabilities.

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 and their lifespan is determined by several timestamps. 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.

The Web service view on Web service data models provides a similar result compared to the human view. The investigation of the UDDI data model exhibits the same weaknesses illustrated in the human view. The lack of search capabilities limits the usability of a Web service view. However, in contrast to the human view, the expressiveness is better, because humans need more descriptive information. UDDI provides the possibility to follow tModel links to external descriptions that include Web service descriptions. The use of UUID for the identification provides also–from the Web service perspective—a powerful identification mechanism. The lack of semantic metadata makes it virtually impossible to automate the discovery process.

The ebXML solution provides better expressiveness regarding the query language in comparison with the UDDI approach. The filtering system provides a simple and extensible selection mechanism that allows the efficient retrieval of Web service descriptions.

The tuple based data model of 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.

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

Table 5 summarizes the view based comparison of the human and Web service view on Web service registry data models.

Table 5. View based Comparison of Web service registry data models. Italics mark differences of the human view compared with the Web service view.

Attributes	UDDI	ebXML	WSDA	WSIL
Expressiveness of Query Language	Poor	Good, *Fair*	Good, *Fair*	n.a
Expressiveness of Web service description	Fair	Good, *Fair*	Good, *Fair*	n.a
Extensibility of data model	Poor, *n.a.*	Good, *n.a.*	Good, *n.a.*	Good, *n.a.*
Semantic Meta Data	No, *n.a.*	Yes, *n.a.*	Yes, *n.a.*	Yes, *n.a.*

5. Case study—Managing a film team

The case study presented in this section serves as an illustration 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. This example was given preference over the well-known travel-arrangement example, since managing a film team better demonstrates various requirements presented in Table 1 and illustrates the human and the Web service view in a richer and highly dynamic environment. 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 5 shows a UML class diagram illustrating our case study:

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 experts, and crew members are all persons with particular capabilities which are provided as services.

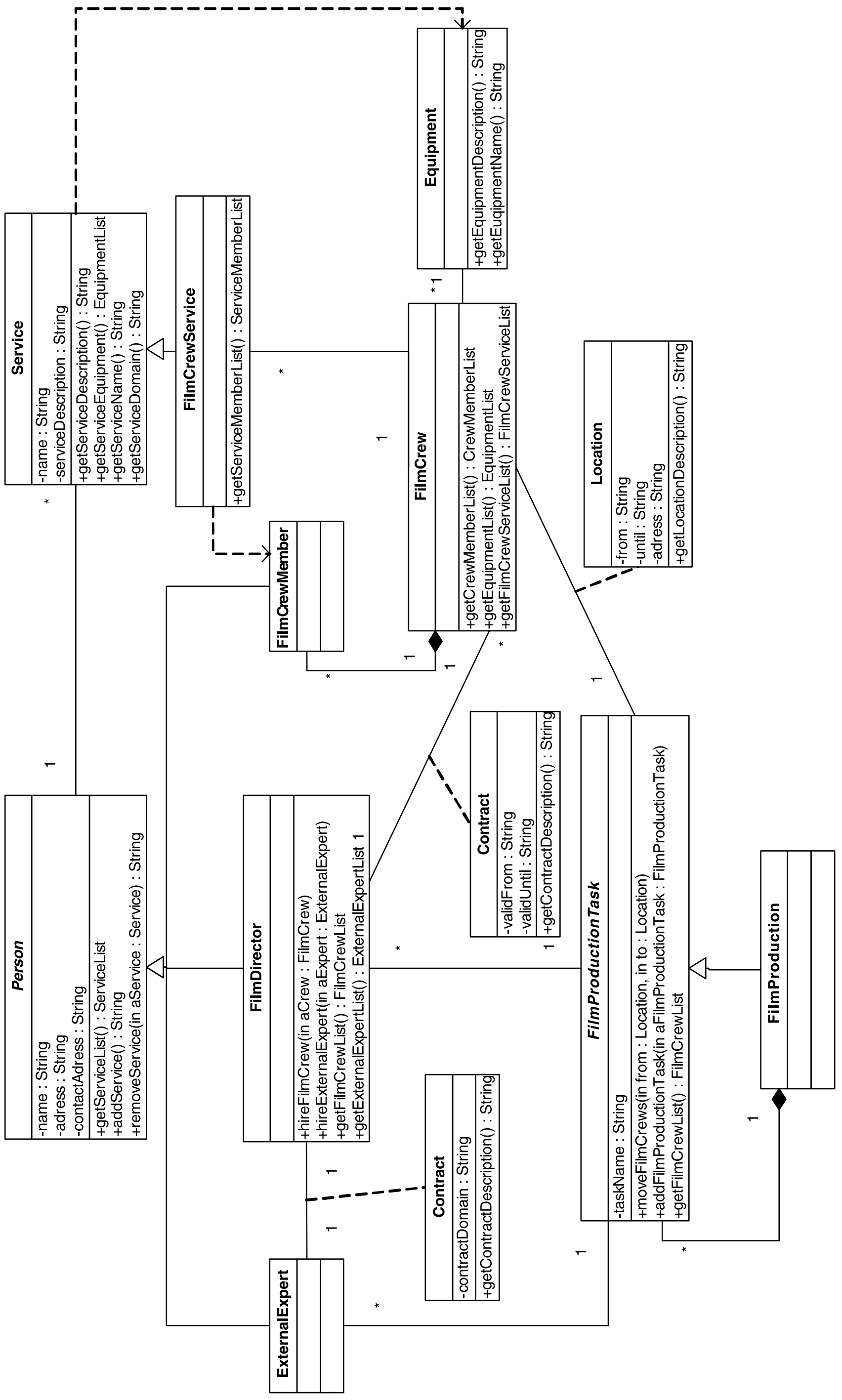
Service
-name : String
-serviceDescription : String
+getServiceDescription() : String
+getServiceEquipment() : EquipmentList
+getServiceName() : String
+getServiceDomain() : String
FilmCrewService
+getServiceMemberList() : ServiceMemberList
Equipment
+getEquipmentDescription() : String
+getEuqipmentName()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

Figure 5. UML class diagram film working example.

Table 6. Film director and Web service view. The columns indicate potential Web service architectures and Web service data models.

Attribute	Film director view	Web service view
Expressiveness of Web service description	ebXML, UDDI	ebXML, UDDI
Expressiveness of Query language	ebXML	ebXML, UDDI, WSDA
Fault tolerance	Hybrid	Hybrid, Decentralized
Security	Hybrid	Hybrid, Centralized
Interoperability	Hybrid	Hybrid
Scalability	Hybrid	Hybrid, Decentralized
Availability	Hybrid	Hybrid, Decentralized

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. In general, 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.

We now apply our view methodology from Section 2. We discuss the view of a film director and the corresponding Web service view to illustrate the differences between both views. We assume that different film crews use different Web service registries respectively are members of different Web service communities. Every film crew is responsible for a certain area of the film-making, for example, a film crew manages the special effects. We consider the film director's view as rather abstract, because the coordination of the different film teams does not involve technical issues. Planning and coordination of the film crews needs a Web service registry architecture that provides simultaneous access to different registries respectively communities. Other concerns regard the expressiveness of Web service descriptions, security, and fault tolerance of the Web service registry architecture.

Table 6 provides the film director's (human view) and the Web service view and shows potential candidates for the Web service data model and candidates for the implementation of the Web service architecture.

6. Conclusion

The two different views on Web service registries share common requirements but exhibit also differences regarding the actual abstractions. For example, both views need expressive

Web service descriptions but depend on different characteristics. Generally speaking, the human view needs informal descriptions and the Web service view depends on well structured metadata. 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 Web service view, semantic markup languages such as DAML+OIL enrich registries and provide meaningful information. Ontologies for arbitrary content like DAML-S [5] 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 [10] 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 ontologies. 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 other registries ontologies to obtain a combined ontology thus implementing relationships between services of different registries.

Another approach is taken in [6]. Directory information is enriched by context aware data which is represented by a Multidimensional OEM graph [18]. This data structure allows to model different facets under different contexts thus providing a hierarchical structure.

7. Future work

An interesting extension to current Web service descriptions could be a common Web service description that unifies the human and Web service view regarding the proposed requirements. This would require a common data model for the description of Web services. This data model should provide the possibility to extract the needed information, respectively provide adequate transformation mechanisms that transform the information according the different–human or Web service–views.

Another issue which is not well dealt with in current Web service registry architectures is the support for transient Web services, i.e., Web services that exhibit unpredictable availability. With the exception of WSDA that provides a tuple space based data model, none of the presented architectures provide efficient support for Web services that exhibit unpredictable availability. The tuple space model seems a promising approach for dynamic Web service registries, since the tuple space concept provides the needed flexibility for dynamic Web service registry entries without much administrative overhead (registering and un-registering). In our future work we plan to provide unified views and mappings between human views and Web service views.

Appendix A

In the following sections we provide examples that illustrate the different Web service registry data models using our case study presented in Section 5.

A.1. UDDI

The example below illustrates a businessEnity structure for our case study 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 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 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>
```

A.2. ebXML

The following example shows our case study 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>
```

A.3. WSDA

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>
```

A.4. WSIL

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>
```

References

1. K. Ballinger, P. Brittenham, A. Malhotra, W.A. Nagy, and S. Pharies, Web services Inspection Language (WS-Inspection) 1.0. IBM, Microsoft, 2001.
2. B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H.H. Ngu, “Declarative composition and peer-to-peer Provisioning of dynamic web services”, in Proceedings of the 18th International Conference on Data Engineering (ICDE’02), IEEE Computer Society, 2002.
3. B. Benatallah, M. Dumas, and Q.Z. Sheng, “Faciliating the rapid development and scalable orchestration of composite Web services”, Distributed and Parallel Databases, vol 17, pp 5–37, 2005.
4. Business Process Specification Schema. http://www.ebxml.org/specs/ebBPSS.pdf, 2001
5. DAML-S Coalition, “DAML-S: Web services description for the semantic Web”, in Proceedings of the International Semantic Web Conference (ISWC), 2002.
6. C. Doulkeridis, E. Valavanis, and M. Vazirgiannis, “Towards a context-aware services directory”, in Proceedings of the 4th VLDB Workshop on Technologies for E-Services (TES ’03), 2003.
7. ebXML Collaboration-Protocol Profile and Agreement Specification. http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf, 2002
8. 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.
9. Hypertext Transfer Protocol — HTTP/1.1. IETF RFC 2616. UC Irvine, Digital Equipment Corporation, MIT, 1999.
10. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, “EDUTELLA: A P2P networking infrastructure based on RDF”, Proceedings of the Eleventh International Conference on World Wide Web, 2002.

11. OASIS/ebXML registry Information Model v2.0. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf, 2001.
12. OASIS/ebXML Technical Architecture Specification. http://www.ebxml.org/specs/ebTA.pdf, 2001.
13. OASIS/ebXML registry services Specification v2.5. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf, 2003.
14. M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara, In Web Services, E-Business and Semantic Web Workshop, 2002, To Appear
15. M.P. Papazoglou, B.J. Krämer, and J. Yang, "Leveraging Web-services and Peer-to-Peer Networks", in Proceedings of the 15th Conference on Advanced Information Systems Engineering (CAiSE '03), 2003.
16. C. Schmidt, and M. Parashar, "A peer-to-peer approach to Web services discovery", in Proceedings of the 2003 International Conference on Web Service (ICWS '03), 2003.
17. Q.Z. Sheng, B. Benatallah, Y.Q. Zhu, R. Stephan, and E.O-Y. Mak, "Discovering E-services Using UDDI in SELF-SERV", in Proceedings of International Conference on e-Business (ICEB2002), 2002.
18. 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.
19. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord:A scalable peer-to-peer lookup services for internet applications", in Proceedings of ACM SIGCOMM'01, 2001.
20. The Web Service Discovery Architecture. Wolfgang Hoschek, in Proc. of the Int'l. IEEE/ACM Supercomputing Conference (SC 2002), IEEE Computer Society Press, November 2002.
21. A. Tsalgatidou and T. Pilioura, "An overview of standards and related technology in Web services", Distributed and Parallel Databases, vol 12, pp 135–162, 2002. Kluwer Academic Publishers. 2002
22. UDDI Version 2.03 Data Structure Reference. http://uddi.org/pubs/DataStructure_v2.htm, 2002.
23. UDDI Version 3.0.1 http://uddi.org/pubs/uddi_v3.htm, 2003.
24. Universal Description, Discovery and Integration: UDDI Technical White paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, 2000.
25. 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.
26. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of Web services", Journal of Information Technology and Management, 2004.
27. W3C. XML Extensible Markup Language. http://www.w3c.org/XML, 2000.
28. W3C, WSDL, Web services Description Language. http://www.w3.org/TR/2002/WD-wsdl12-20020709/, 2002.
29. W3C, SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 2003.
30. W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/2003/WD-xquery-20031112/, 2003.
31. W3C, Web Services Architecture. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2004.
32. W3C, Web Services Architecture Requirements. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2004
33. Wolfgang Hoschek. Peer-to-Peer Grid Databases for Web service Discovery. CERN IT Division, 2002.