Technical University of Vienna
Information Systems Institute
Distributed Systems Group

A Survey on Web Services Composition

Schahram Dustdar and
Wolfgang Schreiner
dustdar@infosys.tuwien.ac.at
e9902261@student.tuwien.ac.at

TUV-1841-2004-15 July 19, 2004

Due to their heterogeneous nature, which stems from the definition of several XML-based standards to overcome platform and language dependence, Web Services have become an emerging and promising technology to design and build complex inter-enterprise business applications out of single Web-based software components. To establish the existence of a global component market in order to enforce extensive software reuse, service composition experienced increasing interest in doing a lot of research effort. This paper discusses the urgent need for service composition, required technologies to perform service composition, presents several different composition strategies, based on some currently existing composition platforms and frameworks representing first implementations of state-of the-art technologies and gives an outlook to essential future research work in the area of service composition.

Keywords: Web Services, Coordination, Transaction, Context, Conversation Modeling, Service Execution, Quality of Service, Service Composition

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 Survey on Web Services Composition

Schahram Dustdar, Wolfgang Schreiner

Distributed Systems Group, Vienna University of Technology
{dustdar@infosys.tuwien.ac.at | e9902261@student.tuwien.ac.at}

Abstract. Due to their heterogeneous nature, which stems from the definition of several XML-based standards to overcome platform and language dependence, Web Services have become an emerging and promising technology to design and build complex inter-enterprise business applications out of single Web-based software components. To establish the existence of a global component market in order to enforce extensive software reuse, service composition experienced increasing interest in doing a lot of research effort. This paper discusses the urgent need for service composition, required technologies to perform service composition, presents several different composition strategies, based on some currently existing composition platforms and frameworks representing first implementations of state-of the-art technologies and gives an outlook to essential future research work in the area of service composition.

Keywords: Web Services, Coordination, Transaction, Context, Conversation Modeling, Service Execution, Quality of Service, Service Composition

1 Introduction

A Web service is a software system identified by a URL, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.
This definition has been published by the World Wide Web Consortium W3C in the Web Services Architecture document [1].

The Web service model consists of three entities, the service provider, the service registry and the service consumer. Other models, such as a peer-to-peer structure, exist as we will discuss later in this paper. Fig. 1 below shows a graphical representation of the traditional Web service model:

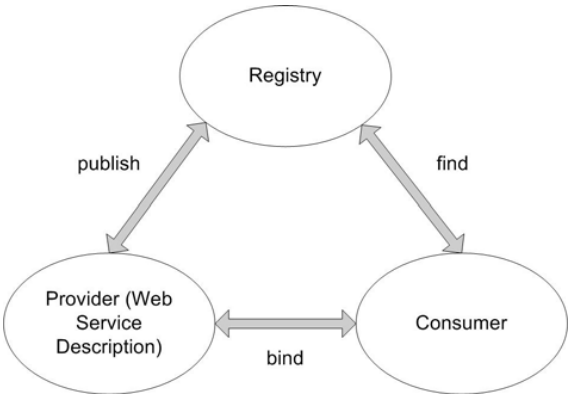

Fig. 1: The Web Service Model

The Service Provider creates or simply offers the Web service. The service provider needs to describe the Web service in a standard format, which in turn is XML and publish it in a central Service Registry. The service registry contains additional information about the service provider, such as address and contact of the providing company and technical details about the service. The Service Consumer retrieves the information from the registry and uses the service description obtained to bind to and invoke the Web service. The appropriate methods are depicted in Fig. 1 by the keywords “publish”, “bind” and “find”. In order to achieve communication among applications running on different platforms and written in different programming languages, standards are needed for each of these operations.

Web services architecture is loosely coupled, service oriented. The Web Service Description Language WSDL uses the XML format to describe the methods provided by a Web service, including input and output parameters, data types and the transport protocol, which is typically HTTP, to be used. The Universal Description Discovery and Integration standard UDDI suggests means to publish details about a Service Provider, the Services that are stored and the opportunity for Service Consumers to find Service Providers and Web service details. Besides UDDI, other standards have been developed as well. [52] deals with Web service registries in greater detail. The Simple Object Access Protocol SOAP is used for XML formatted information exchange among the entities involved in the Web service Model.

Our paper is structured as follows: In Section 2 we will discuss the need for service composition and what advantages we expect from doing research in this area. Section 3 gives a brief overview of our sample scenario that has been developed at our institute and which we will use to describe the different composition strategies we have identified based on the example of managing a film crew. Section 4 contains the main focus of our survey paper. We describe different approaches in service composition by discussing composition platforms and frameworks that have been developed in the course of comprehensive research in the area of service composition. In Section 5 we

try to identify common characteristics and features of the frameworks, by which we can compare them and provide a table as a visual mean to structure our summary and conclusion.

2 The Need for Web Services Composition

The basic Web services infrastructure presented in the previous subchapter suffices to implement simple interactions between a client and a Web service. If the implementation of a Web service's business login involves the invocation of other Web services, it is necessary to combine the functionality of several Web services. In this case we speak of a composite service [2]. The process of developing a composite service in turn is called service composition. Service composition can be either performed by composing elementary or composite services. Composite services in turn are recursively defined as an aggregation of elementary and composite services.
When composing Web services, the business logic of the client is implemented by several services. This is analogous to workflow management, where the application logic is realized by composing autonomous applications.
This allows the definition of increasingly complex applications by progressively aggregating components at higher levels of abstraction. A client invoking a composite service can itself be exposed as a Web service.

Since it is a widely used approach to use conventional programming languages to link components to a composite Web service and thus bridge heterogeneous middleware platforms it becomes necessary to develop a Service Composition Middleware to support composition in terms of abstractions and infrastructure as well [2]. Programming languages focus on APIs rather than on the actual business logic. Different approaches and the need for workflow modeling have finally led to the development of the Business Process Execution Language for Web Services BPEL4WS [20]. A Composition Model and Language to specify the services involved in the composition, a Development Environment with a graphical user interface to drag and drop Web service components and a Run-Time Environment to execute the business logic can be identified as the three main elements of a Web services Composition Middleware. A service composition middleware requires the Web services to be precisely described in their functionality, interfaces and protocols they support. Conventional middleware lacks exactly those features. Workflow Management Systems (WfMS) are highly flexible and generic but on the other hand require the components to be aware of the WFMS API. Hence components are system and vendor-specific and attended to additional development effort.

[2] describes six different dimensions of Service Composition Models. A component model can make different assumptions of what a component is and what it is not. The advantage of a model making very basic assumptions, for example components only have to exchange messages via XML, is a more general model, while it has to deal with much more heterogeneity of the components.

An orchestration model defines abstractions and languages to define the order in which and the conditions under which Web services are invoked. Orchestration models use process modeling languages, such as UML activity diagrams, Petri-nets, statecharts, rule-based orchestration, activity hierarchies and π-calculus. Data and data access models define how data is specified and exchanged between components. The service selection model deals with static and dynamic binding that is how a Web service is selected as a component statically at design-time or dynamically during run-time. Transactions define which transactional semantics can be associated to the composition and how this is done. Finally, we must also consider a model for exception handling to handle exceptional states during the execution of the composite service without the service being aborted.

The Business Process Execution Language for Web Services (BPEL4WS) standard proposed by IBM, Microsoft and BEA is in many cases related to former standards, such as WSFL or XLANG. BPEL descriptions are XML documents, which describe the roles involved in the message exchange, supported port types and orchestration and correlation information as aspects of a process. BPEL4WS is a service composition model which both supports composition and coordination protocols and consists of an activity-based component model, an orchestration model allowing the definition of structured activities, XML schema data types, a service selection model and a mechanism for exception, event and compensation handling.

3 Managing a Film Crew – A Case Study

According to [52] we would like to pick up the case study of managing a film crew in order to explain the different concepts and frameworks introduced in various papers with a consistent reference scenario that runs through the whole paper.
Basically, a film crew consists of one or more film directors, external experts and crew members. Each of these teams provides particular services and work together in a loosely coupled way providing their expertise on demand while they depend on the services provided by other film teams.
A film director must coordinate the film teams, is responsible for the budget and must enable communication among different teams. Since different phases in the film production place different requirements on the arrangement of the teams each phase requires flexible composition and configuration of the film teams. Film crews and external experts are hired by the directors and provide own services and capabilities necessary for shooting a film. External experts offer their expertise on several topics, such as physics, health or law to make the film and special effects more realistic. Crew members may act as stuntmen for example.

4 Composition Issues

Web service composition is a very complex and challenging task as we will see when discussing different composition approaches in greater detail. Before we can perform Web service composition, we have to form some basics to enable service composition. We identified six different issues that have a large impact on service composition. The following subsections do not claim completeness of composition issues. The in our judgement most important issues may be explained in a few words. We introduce the concepts, some standardization efforts and related research work as well. Sun Microsystems, for example, has proposed standards for coordination, transaction and context and put them together to one comprising standard called WS-CAF (Web Services Composite Application Framework) [29].
We discuss coordination, transaction, context and conversation modeling. When composing services we must consider composite service execution. We differ between centralized and distributed service execution, where we will also discuss frameworks that fall into these categories. In addition to the "traditional" SOA (Service Oriented Architecture) mentioned in the introduction based infrastructure there are also alternatives to that. Here we will deal with peer-to-peer infrastructure and describe a P2P based framework called Peer-Serv and present suggestions to extend the classical Web service model in order to precisely describe Web services and facilitate quality driven service discovery

4.1 Coordination

The basic Web service infrastructure presented in Section 1 satisfies simple interactions and method invocations among interacting entities. When it comes to composing Web services and building complex software systems it is likely that interaction requires coordination of sequences of operations to ensure correctness and consistency. New protocols and abstractions are needed and this is exactly the case of coordination. In order to provide modeling abstractions and simplify Web service development, different standardization efforts have been taken, such as WS-Coordination [22] by IBM or WS-CF by Sun [28].

4.2 Transaction

To add guarantee to the interactions it is of considerable importance to add a transaction protocol to the coordination framework in order to provide short-duration transactions, called atomic transactions, and long running business activities as well. In case of long running activities it is not always possible to ensure ACID (atomicity, consistency, integrity, durability) properties of Web service transactions. Very important to name are the WS-Transaction [23, 24, 25] standard, proposed by IBM and WS-TXM [30] suggested by Sun. WS-Transaction builds upon the WS-Coordination

framework mentioned above since it defines protocols for centralized and peer-to-peer transactions, which both require the existence of coordination.

4.3 Context

The term context is a very vague one, since many different definitions can be found in literature. In terms of Web services we may speak of context as information utilized by the Web service to adjust execution and output to provide the client with a customized and personalized behavior [59]. Context is extensible with new types of information at any time without any changes to the underlying infrastructure. Context may contain information such as a consumer's name, address and current location, the type of client device, including hard- and software, the consumer is using, or all kinds of preferences regarding the communication.
The WS-Context [31] standard which is part of the WS-CAF standard proposed by Sun, specifies context, context sharing and context management.

4.4 Conversation Modeling

[14] introduces a framework that builds on currently existing standards to support developers in defining service models and richer Web service abstractions. The authors identify several abstractions, which can be classified as completion or activation abstractions. Completion abstractions contain compensating operations in case an effect of a so called forward operation has to be cancelled and resource-locking operations that lock resources for a client. Activation abstractions describe implicit and explicit transitions between states. An important type of implicit transitions is timed transitions, which occur automatically after some time. To model transitions multiple properties are used: activation properties describe a transition's triggering features, transaction properties specify a transition's effect on the client state and locking properties reserve certain resources for the requester for a given time. The conversation model facilitates service discovery and dynamic binding, service composition model validation, service composition skeleton generation, analysis of compositions and conversations and conversation model generation.

In [40], the authors propose a specification for designing and analyzing service composition. Peers, modeled as Mealy machines, communicate through asynchronous messaging and maintain a queue for incoming messages each.

Related work in Web service conversation modeling include the WSCL (Web Services Conversation Language) specification, the WSCI (Web Service Choreography Interface) specification and also WS-Coordination and WS-Transaction although they do not provide means to model business conversations but rather propose specific conversations that can be used to coordinate interacting parties and provide middleware properties [14].

4.5 Execution Monitoring

We distinguish between centralized and distributed execution of composite Web services. Centralized execution is similar to the client-server paradigm. In this case, the server is the central scheduler that controls the execution of the components of the composite Web service. The eFlow platform developed by HP described in the previous section works with a centralized scheduler [4].
The distributed paradigm in contrast expects the participating Web services to share their execution context. Each of the hosts running a Web service has its own coordinator which has to collaborate with the coordinators of the other hosts to guarantee a correct ordered execution of the services. SELF-SERV [55] developed by the University of New South Wales uses such a distributed execution system. A hybrid form of the distributed and centralized paradigms may be coordinators that control not only one but a set of Web services.

4.6 Infrastructure

[5] suggests another model for discovering Web services based on QoS constraints by extending the basic Web services model discussed in Section 1. 48% of the UDDI entries are unusable. Pointers are missing, broken or contain inaccurate information. Furthermore, service discovery in UDDI is limited to functional requirements only. The approach taken in [5] does not touch the existing Web service model but extends it by adding another entity called the “QoS certifier”. The process of publishing, finding and binding to a Web service remains the same. But it adds the role of the QoS certifier to verify the service provider’s QoS claims before the registration of a service in the UDDI registry can take place. In the new model, a service provider also has to provide QoS information, not just functional aspects. The certifier then approves the claim of the provider or may down grade it.
When a service consumer looks for a service, it may add some QoS constraints to the functional requirements and thus enforce a much finer search. Since there may be many services that meet the functional requirements of the user, the QoS constraints may help to filter the proper services. If no service is found, the user may also consider trade-offs by reducing the QoS constraints.

In order to realize the proposed extension of the Web service model, it is necessary to extend the UDDI data structure by an additional element called “qualityInformation”. The structure provides the description of different QoS aspects, such as availability, reliability or performance. Like the “bindingTemplate” element it refers to the “tModel” defined in the UDDI registry as references to QoS taxonomies. If we now consider a simple SOAP request to a UDDI registry, then it may now, related to the new model, contain QoS parameters to find the proper services that meet these requirements:

```
<SOAP-ENV:Envelope>
       <SOAP-ENV:Body>
              <find_service>
```

```
                    ...
                           <qualityInformation>
                           <availability>0.9</availability>
                           </qualityInformation>
                    </find_service>
             </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

The registry may respond with service references matching the search criteria:

```
      <SOAP-ENV:Envelope>
      <SOAP-ENV:Body>
             <serviceList>
                    <serviceInfos>
                    <serviceInfo serviceKey="…">
                           <qualityInformation>
                           <availability>0.99</availability>
                           </qualityInformation>
                    </serviceInfo>
                    <serviceInfo serviceKey="…">
                           <qualityInformation>
                           <availability>0.91</availability>
                           </qualityInformation>
                    </serviceInfo>
                    </serviceInfos>
             </serviceList>
      </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

In [5] the author organizes different QoS aspects in runtime related QoS, such as scalability, capacity, performance (response time, latency, throughput), reliability, availability, flexibility, exception handling and accuracy, transaction support related QoS, like regulatory, supported standard, stability, cost and completeness and security related QoS, for example authentication, authorization, confidentiality, traceability and auditability, data encryption and non-repudiation.

Apart from conventional Web service model described in Section 1, the idea of a P2P based Web service infrastructure is very common in literature.
P2P systems bring more availability, scalability and extensibility [15]. Peer-Serv is a framework that is composed of several providers, brokers and requestors to support Web services in a P2P environment [15]. Service providers and service requestors have the same role as service providers and service consumers described in the Web service model. Service brokers are responsible to maintain the status of the peers. A service broker maintains providers and requestors as well to form a community. Several communities may then interact with each other over their brokers. Peer-Serv uses P2P technology for service execution as well as for service publishing and service querying.

5 Composition Approaches

We identified five categories of composition strategies. Static and dynamic composition strategies concern the time when Web services are composed. They are equivalent to design-time and run-time composition. We deal with model driven service composition, business rule driven service composition and declarative composition as well. Another topic concerns automated and manual service composition. We provide a survey of ontology based composition and semantic Web services composition as counter-part to manual composition techniques such as provided by BPEL. This section closes with an overview of context based service discovery and composition.

5.1 Static versus Dynamic Service Composition

Static composition takes place during design-time when the architecture and the design of the software system are planned. The components to be used are chosen, linked together and finally compiled and deployed. This may work fine as long as the Web service environment – business partners and service components does not or only rarely change. Microsoft Biztalk and Bea WebLogic are examples of static composition engines [4].
If other businesses provide newer services or the old services are replaced by other ones, inconsistencies might be caused. In that case it is unavoidable to change the software architecture and bind to other services or, in the worst case, even change the process definition and redesign the system. In this case static composition may be too restrictive and components should automatically adapt to unpredictable changes [4]. eFlow from HP or the StarWSCoP both described later in this section implement a dynamic composition engine.
The service environment is a highly flexible and dynamic environment. New services become available on a daily basis and the number of Service Providers is constantly growing. Ideally service processes should be able to transparently adapt to environment changes and to adapt to customer requirements with minimal user intervention.

HP has developed a system called eFlow for specifying, enacting and monitoring composite e-services [3]. HP itself defines e-services as the means by which an enterprise offers its products, services, resources and know-how via the Internet. E-services may be used by persons (business to consumer), enterprises (business to business) and electronic devices and is thus a broader term than e-commerce or e-business.
Composite services are modeled as business processes enacted by a service process engine. eFlow provides features that support service process specification and management, including a simple service composition language, exception handling, ACID transactions and security management. eFlow enables process specifications that automatically configure at run-time, has simple process modification semantics and enforces consistency rules to avoid run-time errors as well as authorization rules.
eFlow runs on top of E-Services Platforms (ESPs), such as HP e-speak or Sun Jini, which basically correspond to the Web service model introduced in Section 1. They

allow the development, deployment and secure delivery of e-services to businesses and customers. Service providers may register service descriptions of services they offer and may monitor and manage service execution. Services may be discovered by Service consumers by specifying search criteria and then be invoked. Service invocation is restricted to authorized users.

In eFlow a composite service is described as a process schema that composes basic or composite services. Composite services are modeled by a graph defining the flow of service invocations similar to UML activity diagrams, which is also translated into a XML structure. eFlow additionally allows to define low-level graphs to specify the order in which certain methods of one specific service are invoked. It also includes "transactional regions" to ensure the atomic execution of parts of the process.

The eFlow engine notifies completions of method nodes, whole service nodes and then determines the next service node to be executed. It is connected to the ESP via the eFlow adapter to get notified whenever the service environment changes, that is, when service definitions change or new providers and services are registered within the ESP. In order to manage and even take advantage of the frequent changes in the service environment, the services need to be adaptive – to adjust to changes in the environment with minimal user intervention. eFlow implements several features, such as dynamic service discovery, dynamic conversation selection, multiservice nodes and generic nodes.

eFlow specifies a service selection rule, which analyzes user requirements to create several input parameters in order to discover appropriate services. If more services are found, the best match is taken. Dynamic conversation selection allows the user to select the conversation within a service node at run-time, which may be useful when services are discovered also at run-time and service interfaces are unknown. Then eFlow may select an appropriate conversation from the conversation repository.

The semantics of multiservice nodes is useful, when it is necessary to invoke multiple, parallel instances of the same type of service.

If a composite service has been defined, this may be fine for several users, but some may need additional functionality. To cope with these needs, eFlow adds the notion of dynamic service creation by including generic service nodes. A user may select additional services from a service pool, which are then put into a service list and passed to the generic service node in order to configure it. Hence, multiservice nodes allow the activation of several instances of the same service node, while generic nodes allow the dynamic selection of different service nodes.

Adaptive processes should reduce the need for human intervention in the process definition. Nevertheless there might still be cases in which service process modifications are unavoidable to handle exceptions, improve the process or incorporate new laws or policies. eFlow distinguishes between ad hoc changes and bulk changes. Ad hoc changes are usually applied to a single process instance. Either the process schema or the state of the process instance may be modified. Ad hoc changes are usually made in exceptional changes and are not expected to occur a second time. Bulk changes are made when some or even all of the running instances are affected. This may be the case when a service does no longer exist or a service definition is changed. Then running instances will have to be notified of the change in the environment in order to cope with the new situation.

In [4] the Web Services Composition Platform, StarWSCoP (Star Web Services Composition Platform), is introduced. It focuses on dynamic Web services Composition. The composition consists of four steps:

1. Service providers publish their services at a Web service registry.
2. The Service Composition Engine decomposes user requirements into an abstract service and sends a SOAP request to the registry to find the proper services.
3. The Web service registry returns a set of concrete services.
4. The service composition engine sends a SOAP request to the concrete services and binds to them.

StarWSCoP includes several modules: an intelligent system to decompose user requirements into the description of an abstract service, a service registry to provide a Web service repository, a service discovery engine to find proper services that meet user requirements in the service registry, a composition engine that schedules the composite services to be executed in order, a wrapper to achieve interoperability of heterogeneous services of different providers that have been developed separately, a service execution information library to store trace information of composite Web service execution, a QoS estimation to estimate real-time QoS metrics of the composite service and an event monitor to monitor events and notify the composition engine.

To support QoS based dynamic Web services composition, WSDL is extended with QoS attributes, such as time, cost or reliability:

```
<portType name="GetTotalFilmCost">
       <operation name="getTotalFilmCost"
               cost="20"
               time="50"
               reliability="90%">
        ...
       </operation>
</portType>
```

An ontology-based layer is added to UDDI to achieve semantic match for Web services.

A second Wrapper is added to the Composition platform to conceal differences or inconsistencies in data types, security policy or content of Web services. The Wrapper adds a Communication manager to translate messages between different transport protocols like HTTP or SMTP, a Security Manager to traverse firewalls and handle authentication and authorization, a Content Manager to convert between different document representations, a Conversation Manager for conversational differences and a QoS Monitor to monitor QoS metrics of Web services.

Web services composition shares many requirements with business process management [6]. Web services, as well as business processes need to coordinate the sequence of service invocations, manage data flow and manage execution of compositions as

transaction units. The difference to business processes is that Web services are autonomous, heterogeneous units, which makes the implementation of Web services composition harder. Each service provider has its own business rules. [6] introduces a framework called “WebTransact” which provides the necessary infrastructure to build reliable service compositions. WebTransact is composed of a multilayered framework containing a Service Composition Layer, a Service Aggregation Layer, an Integration Layer and a Description Layer shown in Fig. 2 below.

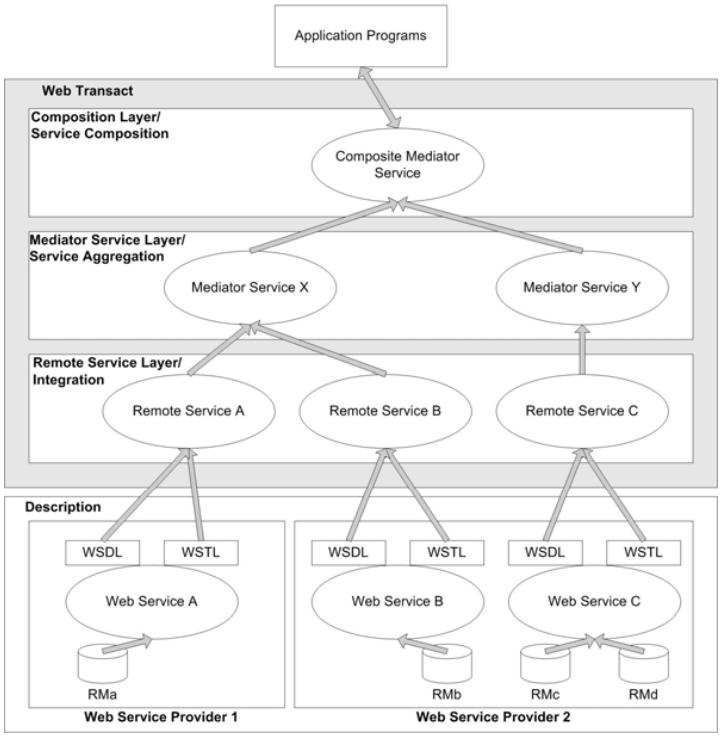

Fig. 2: The multilayered architecture of WebTransact

Starting at the bottom (Description Layer) WebTransact uses WSDL for describing the service functionalities and adds a Web Services Transaction Language WSTL on top of WSDL enhancing it with functionalities facilitating Web services composition. In short, WSTL describes the transaction support for a Web. The code example below shows a WSDL code snippet and the corresponding WSTL extension:

```
<opertation name=”getTotalFilmCost”>
       <input message=”tns:cancelSoapIn”/>
       <output message=”tns:cancelSoapOut”/>
</operation>
```

```
<wstl:transactionDefinitions>
<wstl:transactionBehavior operationName="getTotalFilmCost"
                          type="retriable" />
</wstl:transactionDefinitions>
```

The integration or remote service layer defines a remote service as a logical unit of work that performs a set of operations at a particular site. Each operation has a well-defined transactional behavior. WebTransact defines four types of transaction behavior to accommodate the levels of transaction support: compensable if the effects of an operation after its execution can be undone by invoking another operation, virtual-compensable for operations whose underlying system supports 2PC, retriable if its guaranteed that an operation can succeed after a finite set of repeated executions and pivot if it is neither compensable nor retriable.

The service aggregation or mediator service layer aggregates semantically equivalent remote services providing a homogeneous view of heterogeneous remote services. Semantically equivalent services provide different WSDL descriptions but offer the same functionality. Mediator services are thus simply virtual services delegating method invocations to the corresponding concrete remote service. Operations of mediator services also have a well-defined transactional behavior, which can be either compensable, retriable or pivot. If semantically equivalent remote services have different transactional behaviors, the mediator service will use the least restrictive behavior among the behaviors of the aggregated remote services.
Each mediator service exposes one single interface to make composition easier. A remote service provides mapping information and content description of a Web service to the mediator service. When it comes to composition, a composite mediator service describes the interaction patterns for a set of cooperating mediator service operations needed to complete a certain task.
In WebTransact a composition and the sequence of operations invoked are described by WSTL. WSTL models compositions as composite tasks which can consist of wither atomic tasks or recursively of other composite tasks. Tasks are identified by its signature, execution dependencies, data links and rules. Tasks may also be further subdivided into mandatory or desirable tasks.

5.2 Model Driven Service Composition

The paper [56] introduces the approach of Model Driven Service Composition, which is based on dynamic service composition discussed in the previous section, since it should facilitate the management and development of dynamic service compositions. UML (Unified Modeling Language) is used to provide a high level of abstraction and to enable direct mapping to other standards such as BPEL4WS. The OCL (Object Constraint Language) is used to express business rules and to describe the process flow. Business rules can be used to structure and schedule service composition and to describe service selection and service bindings. The paper identifies two main use cases: the process of service composition development, which may be subdivided into the four phases of service definition, scheduling, construction and execution. In order

to enable representation of all possible service compositions, the paper introduces an information model – an abstract meta-model – which models components and relationships between the components. For that purpose, service composition elements and service composition rules are defined.

Since service composition is closely related to workflow, service composition elements are based on the business process elements activity, representing a well-defined business function, condition, constraining the behavior of the composition by adding pre- and post-conditions, event, describing occurrences during the process of service composition, flow, defining a block of activities, message, containing input- or output-information, provider, describing a party offering concrete services, and the abstract class role, describing a party participating in the service composition process.
Service composition rules can be modeled using OCL and can be structured as follows: structural rules guide the process of structuring, scheduling and prioritizing within service composition, data rules control the use of data and message relations, behavioral rules to control event occurrences and enforce integrity constraints, resource rules to guide the use of resources, such as service selection, providers and event raisers, and, finally, exception rules to guide exceptional behavior in the service composition process.

The business rule driven service composition introduced in [57, 58] is very similar to the model driven service composition. The authors use the same service composition elements and define four components for a composition architecture: the definer, the scheduler, the constructor and the executor to map the four phases of the service composition process. The framework introduced, basically consists of a service composition manager SCM assisting the user in developing, executing and managing service compositions and a service composition repository maintaining composition elements and rules. The SCM determines user requirements and passes them to the definer to define an abstract service composition. In order to find out if any information is already available, the definer prompts the composition engine to query the composition rule repository and in case a corresponding composition rule already exists, the composition elements repository returns the activities to the definer. New rules, on their part, are stored in the composition rule repository and composition elements repository respectively. The scheduler develops a set of concrete compositions and lets the user select an alternative, which is then passed to the constructor to generate the executable software. The executor monitors the service execution.
Paper [57] introduces a classification scheme for business rules: structure related rules facilitate the specification of the way in which service composition is to be carried out, role related rules govern the participants involved in the service composition process, message related rules regulate the use of information, event related rules govern the behavior of service composition in reaction to (un-)expected events and constraint related rules represent conditions in service composition

5.3 Declarative Service Composition

The approach followed by enTish [52] is somewhat different from typical composition platforms. Services are typically created on the fly to realize client requests. Anyway, most frameworks are based on the assumption that first the business process has to be created. For enTish, a different architecture is needed, since client requests are expressed in a declarative way using formal languages. The declarative approach consists of two phases: the first phase takes an initial situation and the desired goal as starting point and constructs generic plans to reach the goal. The latter one chooses one generic plan, discovers appropriate services and builds a workflow out of them. The first phase is realized using PDDL (Planning Domain Definition Language) and estimated-regression planning as used in XSRL (XML Web-services Request Language), which must provide machine readable semantics and specify the abstract service behavior. The second phase may be realized by using existing process modeling languages, such as BPEL.
enTish implements a layered architecture, containing a conversation layer, a functionality layer and a database management layer representing real world data. The functionality layer consists of two components, namely raw application and a filter associated with the raw application. The filter analyzes input information of an operation, which may consist of several parameters, in order to produce the desired output by the raw application. Constraints are expressed in one common description language. The conversation layer implements a conversation protocol specifying the order for message exchange.
Since the architecture is quite different from the traditional Web service architecture presented in the introduction, the concept of Web service description and Web service registry have to be revised. WSDL does not describe service attributes and UDDI lacks of information about what the service does. The goal of the enTish approach is thus to create a description language that overcomes the drawbacks of currently existing standards, expresses client requests and is open and of distributed use to enable users to introduce new resource types, functions and relations with unique names, namely URIs. Furthermore, a coordination model and a universal protocol are required to realize the declarative approach.

In SELF-SERV [55], Web services are declaratively composed and then executed in a dynamic peer-to-peer environment. SELF-SERV defines three types of services: elementary and composite services and service communities. Service communities can be seen as containers of alternative services. Service composition is based on state-charts, glueing together an operation's input- and output-parameters and produced events. Service execution is monitored by software components called coordinators which initiate, control and monitor the state of a composite service they are associated with. The coordinators retrieve the state relevant information from the service's state-chart and represent it in what is called a routing table containing preconditions and postprocessings.
The SELF-SERV framework features a service manager and several pools of services, all implemented in Java and communicating via the exchange of XML documents. The service manager consists of a service discovery engine facilitating the

advertisement and location of services, a service editor facilitating the definition of new services and the edition of existing ones and a service deployer generating routing tables of every state of a service's state-chart and uploading these tables into the hosts of the corresponding composite service. Input and output information are both structured as XML documents.

5.4 Automated versus Manual Web Services Composition

In Section 2, when discussing service composition models, we introduced BPEL4WS as a manual composition model. In this subsection we deal with automated service composition in greater detail. In contrast to manually describing service composition, there is also a lot of effort done in the direction of automated or ontology based service composition. An ontology is a collection of Web services that share the same domain of interest. A Web service dealing with film equipment may belong to a film-production ontology for example. DAML-S (DARPA Agent Markup Language for Web services) provides the mechanisms to organize Web services into ontologies.

[42] deals with the need for the requirements for ontologies. The paper contains a list of some proposed ontologies needed for the management of Web services:

- Ontology of QoS metrics
- Ontology of measurement units
- Ontology of currency units
- Ontology of measured properties
- Ontology of measurement methods

[42] also suggests ontological definition of QoS metrics and summarizes them as follows:

- Metric name
- Short human-readable textual description
- Measured property (a link to the ontology of measured properties)
- Formulae (from zero to many) how the given QoS metric can be computed from other QoS metrics, each accompanied by a unit conversion rule
- Invariant relationships with other QoS metrics

In [9] the Web service environment is characterized by the features exploratory, volatile and dynamic. Exploratory refers to dynamic service invocation during runtime when the need for a certain service is established while the feature dynamic covers the service evolution aspect when the content of a Web service changes over time. Volatile means that a Web service handling a request may not be available at a later point of time. The approach taken here proposes the usage of ontologies in an exploratory service space and the employment of agents to deal with Web service volatility and dynamism.
Changes in Web service execution may be of internal, where referring to the change of information provided by the Web service, or external nature including the volatility of Web services which refers to a temporal or permanent unavailability of a Web

service due to network failure, service relocation, request overload or operation rename. The problem of supporting dynamic service requests contains the two facets Web service discovery and selection and Adapting to Web service changes. The authors define the term change management as generic term for detection, propagation and reaction to internal and external changes as well.
A service request is defined as a list of atomic orders where each in turn consists of application specific attributes. For example stock identifier or stock category in a stock market service environment. Formally, a request may be specified by the following grammar:

```
req      ::=    req ; req |
                req || req |
                order
```

This sample shows that a request may either consist of two sequential requests or of two requests executing concurrently or of an order. An order id defined as tuple containing a value for each attribute necessary for the request. An asterisk may be used as wildcard, if the value can't be specified.

```
Req:   { ("buyEquipment", Camera, *, *, *, 50, 00:15) }
```

The sample request shown in the grey box above describes an order to buy 50 cameras in order to improve the film equipment of the crew. In this example the last value represents a timestamp while the remaining three may take any value.
To overcome the problems that arise when it comes to dynamic service discovery and selection [9] proposes an approach to organize Web services into ontologies.
DAML-S divides service descriptions into profiles, providing a high level description of the service, models describing the execution flow of the service, and groundings providing a mapping between DAML-S and WSDL in order to describe how the service has to be invoked. The profile should contain enough information to actually discover a Web service including functional descriptions of the service such as input or output parameters. The description part provides human understandable information, such as service name and some textual description. The following code snippet gives a brief idea of how a DAML-S description looks like:

```
<daml:Class rdf:ID="Camera">
        <rdfs:label>Equipment Item</rdfs:label>
        <rdfs:subClassOf rdf:resource="&service;"/>
</daml:Class>
<rdf:Property rdf:ID="Equipment">
        <rdfs:label>Equipment</rdfs.label>
        <rdfs:subPropertyOf
                rdf:resource="&profile;serviceCategory"/>
        <rdfs:domain rdf:resource="&service;serviceProfile"/>
        <rdfs:range rdf:resource="&daml;#Thing"/>
```

```
        </rdf:Property>
```

The first four lines represent a human understandable description of the Web service. They define the name and specify that the class is a service. The rest of the code defines a property of the service. If a search is performed, a request is sent to the DAML-S registry which in turn looks for matching services. If a proper description is found, the properties of the service are checked to see if they fit the requirements.
Now, what if the price for a stock for example rises above a certain limit? If a film director wants to buy equipment for a certain amount of money – considering the sample request on the previous page – and the price rises during execution, the request has to be aborted and another service has to be found. To react to changes, [9] suggests the usage of agents that play the roles of monitors and notifiers. Changes may occur before service invocation or during service execution. Soft states store a participant list that is used to maintain the participants in a Web service environment interacting with the Web services. An agent is assigned to each Web service to monitor changes in the status of the service, to verify the availability and to notify the participants. Changes are detected by comparing a service's operations with the corresponding service description in a UDDI registry. If operations have changed, these changes are updated in the registry and the participant list is refreshed by removing references to Web services that have become invalid due to changes of Web service interfaces. The way a participant then reacts to a change depends on the type of change and the availability of alternate services. If no alternate service exists, the request must be cancelled. In case of an internal change the previous data will be replaced by the current response [9].
The authors propose an architecture including a GUI to interact with the user and a request broker as intermediary between GUI and Web services to support dynamic service requests. The GUI allows the user to specify service requests and validates user input. The request broker then handles decomposition of user requirements, service selection, service invocation, change management and response consolidation. The request broker consists of several components which namely are the Communication Interface which is responsible for a secure communication between GUI and Web services using the SOAP standard, the request decomposer which decomposes a request into a list of orders, the request manager which evaluates the request execution and determines whether a request has been executed safely or not and the Notifying sgents which monitor changes in the distributed environment and propagate changes to the entities that need to know about these changes.
Implementation of the system has been done using Java Technologies, such as RMI and JDBC, WSDL and DAML-S to describe the Web services, UDDI to publish service descriptions and SOAP to allow messaging among the entities involved.

[27] proposes a DAML-S process model supporting AI planning techniques for automatic Web service composition called SHOP2. DAML-S and OWL (Web Ontology Language) and DAML+OIL as well are used for semantic Web service description. Web services are seen as actions with pre- and post-conditions. In DAML-S ontology services are modeled as processes: atomic, composite and simple processes. Atomic processes Atomic processes can be directly executed by calling the corresponding

Web service. Composite processes may be decomposed into atomic or other composite processes. Simple processes are abstract representations of atomic or composite processes. They are not directly executable but provide an abstract view of the process.

The DAML-S ontology describes a set of classes and properties, specific to the description of Web services. The upper ontology of DAML-S comprises the service profile for describing service advertisements, the process model for describing the actual program that realizes the service, and the service grounding for describing the transport-level messaging information associated with execution of the program [38]. The code example below shows the definition of the process term in DAML-S:

```
<daml:Class rdf:ID="Process">
  <daml:unionOf rdf:parseType="daml:collection">
  <daml:Class rdf:about="#AtomicProcess"/>
  <daml:Class rdf:about="#SimpleProcess"/>
  <daml:Class rdf:about="#CompositeProcess"/>
  </daml:unionOf>
</daml:Class>
```

The atomic process on the other hand is described as follows:

```
<daml:Class rdf:ID="AtomicProcess">
  <daml:subClassOf rdf:resource="#Process"/>
</daml:Class>
```

To stick with the example described in our case study, we would like to provide a small code sample to give a basic idea of what a DAML-S description of a process definition looks like:

```
<daml:Class rdf:ID=”ShootingLocator”>
  <rdfs:subClassOf rdf:resource=”Process.daml#AtomicProcess”/>
</daml:Class>

<rdf:Property rdf:ID="LocationPreference">
  <rdfs:subPropertyOf rdf:resource="…"/>
  <rdfs:domain rdf:resource="#ShootingLocator"/>
  <rdfs:range rdf:resource="concepts.daml#LocationPreference"/>
</rdf:Property>
```

The class ShootingLocator may be used to find proper locations for shooting a film. It is supposed to be an atomic process and is associated with a LocationPreference property.
SHOP2 [27] is a domain-independent HTN (Hierarchical Task Network) planning system that creates plans by task decomposition. SHOP2 decomposes a task into very small primitive tasks that can be directly executed in the order planned by SHOP2.

The approach taken in [38] describes a framework called OntoMat-Service, which allows for seamlessly browsing conventional web pages, including XHTML advertisements for web services, direct, manual invocation of an advertised web service as a one-off use of the service, tying web service advertisements to each other when browsing them, tying web service advertisements to one's own conceptualization of the web space when browsing them and invoking such aggregated web services [38]. The WSDL definition listed below shows the typical WSDL elements `types`, `message` and `portType` with one or more `operation` tags enclosed by the `definitions` root element. The `types` element is enhanced by RDF class and properties definitions. The RDF part describes a class called `Filmcrew` in our example with a property element further specifying the service class.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="LaptopService"
             targetNamespace="http://filmmanagement.wsdl/"
  <types>
    <rdf:RDF>
      <rdfs:Class rdf:ID="Filmcrew">
        <rdfs:label>Film crew</rdfs:label>
      </rdfs:Class>
      <rdf:Property rdf:ID="wage">
        <rdfs:label>Wage</rdfs:label>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
        <rdfs:domain rdf:resource="#Filmcrew"/>
      </rdf:Property>
    </rdf:RDF>
  </types>
  <message name="totalFilmCost">
    <part name="cost" type="rdf:ID=costs"/>
  </message>
  ...
  <portType name="FilmcrewService">
    <operation name="getTotalFilmCost" />
      <output message="tns:totalFilmCost"
              name="totalFilmCost"/>
    </operation>
    ...
  </portType>
</definitions>
```

The OntoMat-Service starts with a common WSDL service description by the service provider. The WSDL description is then parsed and made human readable and formatted as an HTML document. This page may contain several service descriptions and cross-links to related services. In the next step the user may either choose text-phrases that correspond to some underlying machine understandable semantics and map them to his own terminology or directly invoke the Web service. The result of the mapping step is a set of mapping rules between the Web service ontologies and preloaded ontologies. Before invoking the Web service operations a user might wish to choose the operations and the mappings he wants to use. The OntoMat-Service provides a Web service planning module which computes logically possible service

flows. Therefore a knowledge base provides pre- and postconditions, goals, etc. for Web services.
The OntoMat-Service-Surfer described in the final section represents a graphical user interface which implements a drag and drop mechanism enabling the user to highlight items on a web page and pulling them into the ontology browser of the OntoMat-Service framework. The resulting ontologies may in turn be published to enable third parties to execute services based on these semantics.

In [34] the authors focus on a composability model for Web services, automatic generation of composite Web services and prototype implementation and experiments. They develop a framework called WebDG (Web Digital Government) to provide E-government services and to serve citizens in collecting benefits to which they are entitled especially those facing low income, mental illness, addiction or mental retardation
Web service composition requires semantic service description in order to enable services to interact with each other. WSDL does not provide any semantic description to Web services. The figure below depicts a diagram developed by [34], which shows the key features of WSDL in ovals surrounded by dashed lines and those added by semantic descriptions in grey filled ellipses. Edges are unidirectional and add multiplicity information to describe the relations between the entities.

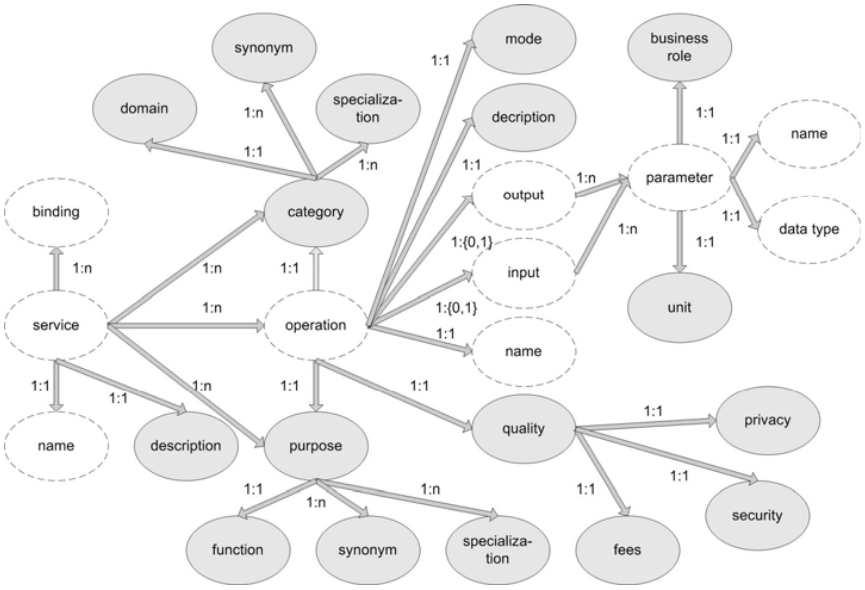

Fig. 3: Ontology based description of Web services

Operations for example consist of an input message, an output message, exactly one name, a description and an operation mode, which may either be notification, solicit response, one-way or request-response. Furthermore it can be described by purpose and quality, which are tuples of three attributes. Function, synonym and specialization define the operation's purpose and fees, security and privacy specify the operation's quality. Operations are categorized by domain, synonym and specialization. Domains are areas of interest where the operation might be used (for example an operation `calculateFilmCost` may only be interesting for a film production company). Synonyms are alternative function names and specialization contains a set of characteristics of a certain function.

After extending WSDL with semantic capabilities it is now possible to deal with a composability model for Web services. [34] identifies two different composability rules: syntactic and semantic rules. Syntactic rules include mode and service composability. Assuming that two services are mode composable – a one way operation at either the client or server side must be connected to a notification mode at the other side and solicit response maps to the request-response messaging pattern – both services must also agree to compatible transport protocols in order to be able to exchange messages.

Semantic rules comprise message composability, operation semantics composability, qualitative composability and composition soundness. Message composability is based on the definition that message types, which in fact are parameter data types, used by the operation to be invoked must be compatible with the ones passed by the client call. Messages are directly compatible if they have the same data type and indirectly compatible if the type used by the client call is derived from the one used by the Web service operation. Operations are semantically composable if the domains of interest are similar or synonymous and both operations provide the same characteristics as well. Operations are qualitative composable if both match the same qualitative properties such as security or privacy. To compose services in a way that provides an added value [34] introduces the notion of composition templates to provide composability soundness. For example two independent services such as `engageFilmCrew` and `bookFlight` may be composed in order to engage actors and actresses for shooting a film and flying them to the shooting location. Composition templates are associated with each composite service and are used to compare values added by different compositions. To check whether a composition is sound or not, stored templates are used to store a composite service's template in the service repository. A service composition is considered as sound if it's template is the subset of a stored template.

With all these rules in mind, [34] proposes an architecture for automatic service composition. The approach consists of four large modules, called Specification, Matchmaking, Selection and Generation. For the service selection phase [34] introduces a high level language – CSSL (Composite Service Specification Language) – which is a semantic description language for Web service compositions. CSSL extends WSDL and represents the ontology based graph presented in Fig.3. In the matchmaking phase a special matchmaking algorithm, integrating the composability rules described above, is applied to generate composition plans that meet the composer's requirements. QoS constraints are used in the selection phase to select the most proper com-

position plan from the output. Finally, the resulting composite service is generated using a composition language such as WSFL, XLANG or BPEL4WS as specified in the CSSL description.

5.5 Context based Web Service Discovery and Composition

Services become available through different channels, meaning that e-services may be accessed using different devices, such as PCs, palmtops, cell-phones or TV sets, and also different network technologies and protocols [7]. The goal should be to deliver the same service via Web, SMS or call centers for example. The Multichannel Adaptive Information Systems project MAIS proposed in [7] aims to create a platform, a methodology and design tools for the development of distributed information systems based on e-services. [7] defines a modeling framework for adaptive information systems separating the application and technological levels and providing formalized contracts between the Service Provider and the Service Consumer.
E-Services in this context are defined by a request perspective and a provisioning perspective, both dealing with QoS issues in order to allow service negotiation based on user requirements, channels and provider constraints. Negotiation is encoded in adaptation rules, which allow the dynamic adaptation of the execution flow of the services, described as workflows.
The provisioning perspective specifies the Service Provides, the service functionality and how to invoke the service. The request perspective specifies the Service Consumer, the QoS level and a certain context. In [7] the authors use UML to describe the entities involved in both perspectives and show its associations.

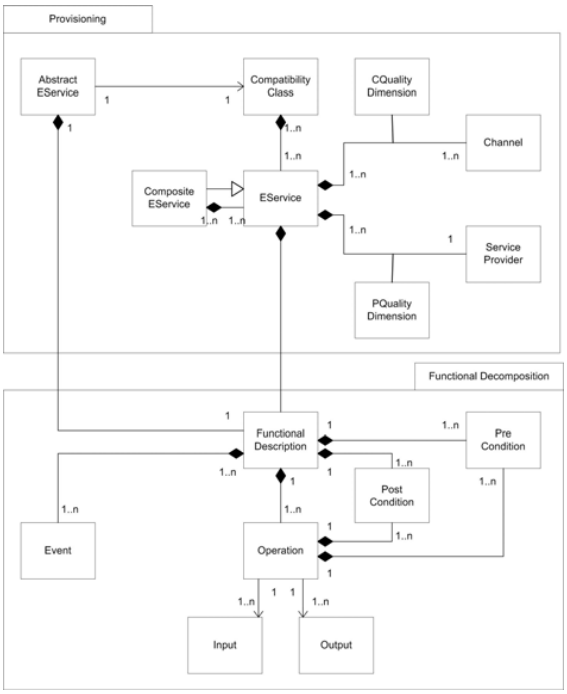

Fig. 4: The provisioning perspective

In the provisioning perspective shown in Fig. 4 above, an EService is described by a unique identifier, a textual description and a category. An EService is associated with one or more Channels and an association class called CQualityDimension to further specify QoS attributes, such as response time, availability, usability, accessibility, integrity, bandwidth, reliability and price. An EService is also associated with exactly one ServiceProvider and an association class called PQualityDimension to describe, similar to CQualityDimension, quality parameters guaranteed by the provider. The FunctionalDescription of an EService contains preconditions, postconditions, operations with inputs and outputs and Events that may occur during the execution of an operation. Finally, an EService is grouped into CompatibilityClasses associated with an AbstractEService to express several services with the same functionality in terms of the process execution to enable service substitution during execution if a service fails. Therefore, an AbstractEService has an association to the FunctionalDescription of an EService. Of course, EServices may be aggregated and composed to a CompositeEService, which extends the EService class.

The class structure depicted in Fig. 4 can be translated into a XML structure as depicted below:

```
<Service>
        <Provider>
                <PQualityDimension>
                <ProvisioningTime>10</ProvisioningTime>
                <Availability>90%</Availability>
                <Price>100</Price>
                </PQualityDimension>
        </Provider>
        <Channels>
                <Channel name="…">
                <Device>PDA</Device>
                <Network>GPRS</Network>
                <CQualityDimensions>
                        <Availability>99%</Availability>
                </CQualityDimensions>
                </Channel>
        </Channels>
</Service>
```

Services may be accessed by different types of devices using different technological channels. Information about networks, devices and service providers is supplied by a context description used by the context manager in the MAIS architecture. The context manager is accessed both by service providers and service consumers as well. The context manager is linked to an interaction enabling platform via an adaptive channel. The interaction enabling platform passes the QoS requests to the above E-Service composition platform, which retrieves the services matching user requirements out of an E-Service directory and performs the composition based on the adaptation rules.

Heterogeneous client capabilities and the increasing number of connected devices require information services to be accessible from all devices in a similar fashion [16]. The authors define the word “context” as the kind of information that makes information services aware of their current context.
They propose a framework in which the context information is passed within a SOAP header between Web Services. The header may contain several context blocks, each associated with one context type. During execution a Web Service may change its context information by inserting different context information into the SOAP header and sending the SOAP message to another Web Service for example. In total there are four components that may process context information: the Web Service, context plug-ins, context services and also clients. Web Services always have full control over the context information. They decide how the information influences their execution and their replies. Context plug-ins are programmed in Java and installed at each local host. Each plug-in is associated with one context type. Context services are also associated with one context type and must be available over the Internet. It is not necessary to install them locally. Context information is preprocessed and post-processed before actually sent by the Web service. In the framework the context block could be processed by several components. Thus processing instructions are needed to specify rules of precedence.
The context framework proposed here provides several context types, such as location (GPS coordinates, country, local time and time zone), information about the

consumer invoking a service (name, email address, preferences) and client context information such as hardware or software. The authors demonstrate the functionality of the framework by developing an information service, which is invoked by several different clients, such as PCs or PDAs. In case of a PDA invoking the service, unnecessary representation information is cut out of the SOAP message since the display size would not meet the graphical representation requirements.

Accessing data via mobile or pervasive systems through Web services has the advantages of hiding the heterogeneous nature of data and providing a well-defined interface for data access [20]. In such a dynamic environment the service discovery plays a crucial role. The approach taken in this paper is to describe a Web Service entry in a Web Service registry in a context-based way that answers the following questions:

- Which services (or devices) are available in location L at time T?
- Which services return results that the requestor device can represent?
- Which services were available at timestamp T?
- Which services are published by user U and his device D?

Since nowadays applications rarely use context information, the aim of the paper is to provide a context representation model to handle any type of context information as long as it can be represented as key-value pairs. In order to develop a context-aware service directory, the authors introduce a data model called Multidimensional OEM (MOEM) graph, which may hold information presenting different facets under different contexts.

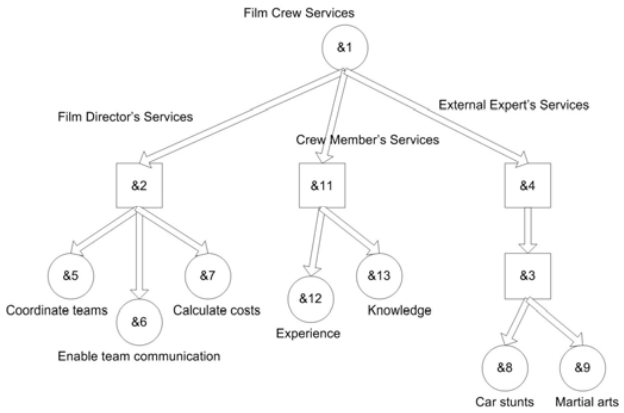

Fig. 5: A MOEM graph representing a service directory

Fig. 5 shows the graphical representation of a service directory as a MOEM graph. Each service category may be represented as an individual MOEM graph. Rectangular nodes represent multidimensional entities that present different facets under different contexts. Circular nodes may either contain data or hold the reference to a service. Bold lines are called context edges and define the context under which the services hold. The plain entity edges represent relations between the entities.

To search and find services matching a certain search criteria, a breadth-first search algorithm is developed. It takes the user defined context description as input and performs the search starting at the root. If a context edge matches the context specifier, the node is kept for further processing, else it is ignored. The result of the search is a list of all services which meet the search criteria.
The MOEM graph is similar to a tree with the exception that a leaf can belong to one or more subtrees. Subsequently an alternate data representation is introduced by translating the MOEM graph into an equivalent graph representation and by the means of a state table containing the device-type, user, return type and service identifier in order to improve the search.

6 Summary

So far, we have dealt with many different approaches and frameworks that have been developed in order to provide widely usable Web service composition platforms. The aim of this section is to give a summary of what we have presented so far. We want to achieve this by creating a table listing most of the frameworks we have discussed within this paper, mapping them to the corresponding composition approach and listing some of the most important features we identified to categorize the composition frameworks. Some approaches, such as enTish do not meet the requirements listed in the table below, since they do not provide a development platform for composite Web services but rather propose language or protocol specifications.

By the analysis of the composition frameworks we have identified some modules and support features that are of great importance for designing and developing composite services. Those include an execution monitor to monitor and trace service execution, dynamic service selection, which is a profound part of service composition and may be used to choose appropriate service conversation protocols from a conversation repository, QoS modeling and evaluation, transaction support and the possibility to graphically modeling service flows.

Framework	**Composition strategy**	**Execution monitor**	**Dynamic conversation selection**	**QoS Modeling**	**WSDL Language extension**	**Transaction support**	**Graph support**
e-Flow	Dynamic composition	yes	yes	no	no	no	yes
MAIS	Context	no	no	yes	yes	no	no

	based						
MOEM	Context based	no	no	no	no	no	yes
SELF-SERV	Declarative composition	no	no	no	no	no	yes
OntoMat-Service	Semantic composition	no	no	no	yes	no	no
SHOP2	Semantic composition	no	no	no	yes	no	no
WebTrans-act	Dynamic composition	no	no	no	yes	yes	no
StarWSCoP	Dynamic composition	yes	no	yes	yes	no	no

7 Conclusion

Many different standards have been proposed and various approaches have been taken to create a widely accepted and usable service composition platform for developing composite Web services. Web services composition seems to have higher chances of success compared to traditional composition middleware due to the standardization efforts that have taken place already. With WSDL, Web services may be described in a consistent way according to their functionalities Of course, QoS and semantic descriptions have been proposed to extend the current WSDL standard, but have not yet found overall acceptance. UDDI and other registry based data models have been implemented, but are not widely used and in case of dynamic service discovery it does not yet meet the requirements.

In this paper we have presented different service composition strategies, introduced some existing composition platforms and frameworks and made an attempt to compare existing research approaches with each other by finding common characteristics and features.

New software is increasingly built by Web services. Hence, service composition is a crucial topic which has a high impact on many different areas of research. This is our main ambition to enforce our research interest in the service composition area and also service evolution since it is very likely that aggregated services may be changed over time.

We have to consider specifications, interactions, non functional attributes and internal changes invisible to the outside world. We must provide different views on service changes, who is managing the changes and who propagates them, issue topics, such as service metrics and monitoring of service execution and service environment. Additionally we must think of disruptive changes when we want a Web service to

cooperate with others, constructive changes when we want to provide better services than those offered so far. The latter topic is closely related to planned changes while unplanned changes must be made whenever a service fails. All these considerations affect issues such as compatibility and versioning of Web services. In our ongoing research work in service composition and evolution, we expect a reduction of development time of services, integration effort and testing, provisioning an integration platform for management and monitoring of aggregated services and their environments and methodologies for testing dynamic service interactions and attributes.

References

[1] Booth, David, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris & David Orchard: *Web Services Architecture. W3C Working Group Note 11 February 2004*. In: W3C Technical Reports and Publications. http://www.w3.org/TR/ws-arch/

[2] Alonso, Gustavo, Fabio Casati, Harumi Kuno & Vijay Machiraju: *Web Services. Concepts, Architectures and Applications*. Springer-Verlag Berlin Heidelberg 2004

[3] Casati, Fabio & Ming-Chien Shan: *Dynamic and adaptive composition of e-services*. 2001 Published by Elsevier Science Ltd.

[4] Sun, Haiyan, Xiaodong Wang, Bin Zhou & Peng Zou: *Research and Implementation of Dynamic Web Services Composition*. APPT 2003, LNCS 2834, pp. 457-466, 2003. Springer-Verlag Berlin Heidelberg 2003

[5] Ran, Shuping: *A Model for Web Services Discovery With QoS*. ACM 2003

[6] Pires, Paulo F., Mario R.F. Benevides & Marta Mattoso: *Building Reliable Web Services Compositions*. Web Databases and Web Services 2002, LNCS 2593, pp. 59-72, 2003. Springer-Verlag Berlin Heidelberg 2003

[7] Baresi, L., D. Bianchini, V. De Antonellis, M.G. Fugini, B. Pernici & P. Plebani: *Context-Aware Composition of E-services*. TES 2003, LNCS 2819, pp.28-41, 2003. Springer-Verlag Berlin Heidelberg 2003

[8] Casati, Fabio, Ski Ilnicki, Li-Jie Jin & Ming-Chien Shan: *An Open, Flexible, and Configurable System for Service Composition*. HPL technical report HPL-2000-41

[9] Akram, Mohammad Salman, Brahim Medjahed & Athman Bouguettaya: *Supporting Dynamic Changes in Web Service Environments*. ICSOC 2003, LNCS 2910, pp. 319-334, 2003. Springer-Verlag Berlin Heidelberg 2003

[10] Benatallah, B., M. Dumas, M.-C. Fauvet & F.A. Rabhi: *Towards Patterns of Web Services Composition*. November 2001

[11] Khalaf, Rania & Frank Leymann: *On Web Services Aggregation*. B. Benatallah & M.-C. Shan (Eds.): TES 2003, LNCS 2819, pp. 1-13, 2003. Springer-Verlag Berlin Heidelberg 2003

[12] Benatallah, B., M. Dumas, M.-C. Fauvet, F.A. Rabhi & Quan Z. Sheng.: *Overview of Some Patterns for Architecting and Managing Composite Web Services*. ACM SIGecom Exchanges, Vol. 3, No. 3, August 2002, Pages 9-16

[13] Papazoglou, Michael P.: *Web Services and Business Transactions*. World Wide Web: Internet and Web Information Systems, 6, 49-91, 2003. 2003 Kluwer Academic Publishers.

[14] Benatallah, Boualem, Fabio Casati & Farouk Toumani: *Web Service Conversation Modeling. A Cornerstone for E-Business Automation*. IEEE Internet Computing, January, February 2004

[15] Wang, Qing et al.: *Peer-Serv: A Framework of Web Service in Peer-to-Peer Environment*. G. Dong et al.: WAIM 2003, LNCS 2762, pp. 298-305, 2003. Springer-Verlag Berlin Heidelberg 2003

[16] Keidl, Markus & Alfons Kemper: *A Framework for Context-Aware Adaptable Web-Services*. E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 826-829, 2004. Springer-Verlag Berlin Heidelberg 2004

[17] Hansen, Mark, Stuart Madnick & Michael Siegel: *Process Aggregation Using Web Services*. Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 12-27, 2002. Springer-Verlag Berlin Heidelberg 2002

[18] Benatallah, Boualem et al.: *On Automating Web services discovery*. Edited by V. Atluri. Received: December 15, 2002 / Accepted: September 15, 2003. Published online: February 6, 2004. Springer-Verlag 2004

[19] Álvarez, P. et al.: *A Java Coordination Tool for Web-Service Architectures: The Location-Based Service Context*. N. Guelfi et al. (Eds.): FIDJI 2002, LNCS 2604, pp. 1-14, 2003. Springer-Verlag Berlin Heidelberg 2003

[20] Doulkeridis, Christos, Efstratios Valavanis & Michalis Vazirgiannis: *Towards A Context-Aware Service Directory*. B. Benatallah and M.-C. Shan (Eds.): TES 2003, LNCS 2819, pp. 54-65, 2003. Springer-Verlag Berlin Heidelberg 2003

[21] Andrews, Tony et al.: *Specification: Business Process Execution Language for Web Services Version 1.1*. In: http://www-106.ibm.com/developerworks/library/ws-bpel/

[22] Cabrera, Luis Felipe et al.: *Web Services Coordination (WS-Coordination)*. In: ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf

[23] Cabrera, Felipe et al.: *Specification: Web Services Transaction (WS-Transaction)*. In: http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/

[24] Freund, Tom & Tony Story: *Transactions in the world of Web Services, Part 1. An overview of WS-Transaction and WS-Coordination*. In: http://www-106.ibm.com/developerworks/webservices/library/ws-wstx1/

[25] Freund, Tom & Tony Story: *Transactions in the world of Web Services, Part 2. An overview of WS-Transaction and WS-Coordination*. In: http://www-106.ibm.com/developerworks/webservices/library/ws-wstx2/

[26] Virdell, Margie: *Business processes and workflow in the Web services world*. In: http://www-106.ibm.com/developerworks/webservices/library/ws-work.html. January, 2003

[27] Wu, Dan, Bijan Parsia, Evren Sirin, James Hendler & Dana Nau: *Automating DAML-S Web Services Composition Using SHOP-2*

[28] Bunting, Doug, Martin Chapman Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber & Keith Swenson: *Web Services Coordination Framework (WS-CF) Ver1.0*. In: http://developers.sun.com/techtopics/webservices/wscaf/wscf.pdf. July 28, 2003

[29] Bunting, Doug, Martin Chapman Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber & Keith Swenson: *Web Services Composite Application Framework (WS-CAF) Ver1.0*.
In: http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf. July 28, 2003

[30] Bunting, Doug, Martin Chapman Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber & Keith Swenson: *Web Services Transaction Management (WS-TXM) Ver1.0*.
In: http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf. July 28, 2003

[31] Bunting, Doug, Martin Chapman Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber & Keith Swenson: *Web Services Context (WS-Context) Ver1.0*. In: http://developers.sun.com/techtopics/webservices/wscaf/wsctx.pdf. July 28, 2003

[32] Georgakopulous, Dimitrios, Hans Schuster, Andrzej Cichocki & Donald Baker: *Process-Based E-Service Composition for Modeling and Automating Zero Latency Supply Chains*. Information System Frontiers 4:1, 33-54, 2002. 2002 Kluwer Academic Publishers

[33] Preuner, Günter & Michael Schrefl: *Integration of Web Services into Workflows through a Multi-Level Schema Architecture*. Proceedings of the 4th IEEE Int'l Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002). 2002 IEEE

[34] Su, Stanley Y.W., Jie Meng, Raja Krithivasan, Seema Degwekar & Sumi Helal: *Dynamic Inter-Enterprise Workflow Management in a Constraint-Based E-Service Infrastructure*. Electronic Commerce Research, 3: 9-24 (2003). 2003 Kluwer Academic Publishers

[35] Medjahed, Brahim, Athman Bouguettaya & Ahmed K. Elmagarmid: *Composing Web Services on the Semantic Web*. The VLDB Journal (2003) 12: 333-351. Published online: September 23, 2003. Springer-Verlag 2003

[36] Cardoso, Jorge & Amit Sheth: *Semantic E-Workflow Composition*. Journal of Intelligent Information Systems, 21:3, 191-225, 2003. 2003 Kluwer Academic Publishers

[37] Terai, Koichi, Noriaki Izumi & Takahira Yamaguchi: *Coordinating Web Services based on Business Models*. ICEC 2003, Pittsburgh, PA. ACM 1-58113-788-5/03/09

[38] Zeng, Liangzhao, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam & Quan Z. Sheng: *Quality Driven Web Services Composition*. WWW 2003, May 20-24, 2003, Budapest, Hungary. ACM 1-58113-680-3/03/0005

[39] Agarwal, Sudhir, Siegfried Handschuh & Steffen Staab: *Surfing the Service Web*. D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 211-226, 2003. Springer-Verlag Berlin Heidelberg 2003

[40] Bultan, Tevfik, Xiang Fu, Richard Hull & Jianwen Su: *Conversation Specification: A New Approach to Design and Analysis of E-Service Composition*. WWW 2003, May 20-24, 2003, Budapest, Hungary. ACM 1-58113-680-3/03/0005

[41] Narayanan Srini & Sheila A. McIlraith: *Simulation, Verification and Automated Composition of Web Services*. WWW 2002, May 7-11, 2002, Honolulu, Hawaii, USA. ACM 1-58113-449-5/02/0005

[42] Tosic, Vladimir, Kruti Patel & Bernard Pagurek: *WSOL – Web Service Offerings Language*. Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 57-67, 2002. Springer-Verlag Berlin Heidelberg 2002

[43] Tosic, Vladimir, Babak Esfandiari, Bernard Pagurek & Kruti Patel: *On Requirements for Ontologies in Management of Web Services*. Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 237-247, 2002. Springer-Verlag Berlin Heidelberg 2002

[44] Horrocks, Ian & Sergio Tessaris: *Querying the Semantic Web: A Formal Approach*. I. Horrocks & J. Hendler (Eds.): ISCW 2002, LNCS 2342, pp. 177-191, 2002. Springer-Verlag Berlin Heidelberg 2002

[45] Motta, Enrico, John Dominigue, Liliana Cabral & Mauro Gaspari: *IRS-II: A Framework and Infrastructure for Semantic Web Services*. D. Fensel et al. (Eds.): ISCW 2003, LNCS 2870, pp. 306-318, 2003. Springer-Verlag Berlin Heidelberg 2003

[46] Wohed, Petia, Will M.P. va der Aalst, Marlon Dumas & Arthur H.M. ter Hofstede: *Analysis of Web Services Composition Languages: The Case of BPEL4WS*. L.Y. Song et al. (Eds.): ER 2003, LNCS 2813, pp. 200-215, 2003. Springer-Verlag Berlin Heidelberg 2003

[47] Velasco, Juan R. & Sergio F. Castillo: *Mobile Agents for Web Service Composition*. K. Bauknecht, A Min Tjoa, G. Quirchmayr (Eds.): EC-Web 2003, LNCS 2738, pp. 135-144, 2003. Springer-Verlag Berlin Heidelberg 2003

[48] Fileto, Renato, Ling Liu, Calton Pu, Eduardo Delgado Assad & Claudia Bauzer Medeiros: *POESIA: An ontological workflow approach for composing Web Services in agriculture*. The VLDB Journal (2003) 12: 352-367. Published online: September 30, 2003. Springer-Verlag 2003

[49] Rao, Jinghai & Xiaomeng Su: *Toward the Composition of Semantic Web Services*. M. Li et al. (Eds.): GCC 2003, LNCS 3033, pp. 760-767, 2004. Springer-Verlag Berlin Heidelberg 2004

[50] Piccinelli, Giacomo & Scott Lane Williams: *Workflow: A Language for Composing Web Services*. W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 13-24, 2003. Springer-Verlag Berlin Heidelberg 2003

[51] Chen, Liming, Nigel R. Shadbolt, Carole Goble, Feng Tao, Simon J. Cox, Colin Puleston & P.R. Smart: *Towards a Knowledge-Based Approach to Semantic Service Composition*. D. Fensel et al. (Eds.): ISCW 2003, LNCS 2870, pp. 319-334, 2003. Springer-Verlag Berlin Heidelberg 2003

[52] Ambroszkiewicz, Stanislaw: *enTish: An Approach to Service Composition*. B. Benatallah & M.C. Shaw (Eds.): TES 2003, LNCS 2819, pp. 168-178, 2003. Springer-Verlag Berlin Heidelberg 2003

[53] Dustdar, Schahram & Martin Treiber: *A View Based Survey on Web services Registries*. Distributed Systems Group, Vienna University of Technologies.

[54] Srivastava, Biplav & Jana Koehler: *Web Service Composition – Current Solutions and Open Problems*. In: http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf

[55] Sheng, Quan Z., Boualem Benatallah, Marlon Dumas & Eileen Oi-Yan Mak: *SELF-SERV – A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment*. Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

[56] Orriens, Bart, Jian Yang & Mike P. Papazoglou: *Model Driven Service Composition*. M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp.75-90, 2003. Springer-Verlag Berlin Heidelberg 2003

[57] Orriens, Bart, Jian Yang & Mike P. Papazoglou: *A Framework for Business Rule Driven Web Service Composition*. M.A. Jeusfeld & Ó. Pastor (Eds.): ER 2003 Workshops, LNCS 2814, pp. 52-64, 2003. Springer-Verlag Berlin Heidelberg 2003

[58] Orriens, Bart, Jian Yang & Mike P. Papazoglou: *A Framework for Business Rule Driven Service Composition*. B. Benatallah & M.-C. Shan (Eds.): TES 2003, LNCS 2819, pp. 14-27, 2003. Springer-Verlag Berlin Heidelberg 2003

[59] Keidl, Markus & Alfons Kemper: *Towards Context-Aware Adaptable Web Services*. WWW2004, May 17-22, 2004, New York, New York, USA. ACM 1-58113-912-8/04/0005.