A Conceptual Architecture for Semantic Web Services Development and Deployment

Claus Pahl

Dublin City University

School of Computing

Dublin 9

Ireland

Abstract. Several extensions of the Web Services Framework (WSF) have been proposed. The combination with Semantic Web technologies introduces a notion of semantics, which can enhance scalability through automation. Service composition to processes is an equally important issue. Ontology technology – the core of the Semantic Web – can be the central building block of an extension endeavour. We present a conceptual architecture for ontology-based Web service development and deployment. The development of service-based software systems within the WSF is gaining increasing importance. We show how ontologies can integrate models, languages, infrastructure, and activities within this architecture to support reuse and composition of semantic Web services.

Keywords. Web Services, Semantic Web, ontology technology, semantics, service processes, conceptual development and deployment architecture.

1 Introduction

The Web Services Framework (WSF) [1] aims at opening the Web for software applications. Services are self-contained computational entities, made available through the infrastructure provided by a provider and used as is by service requesters.

The focus of the WSF core is on the boundaries of systems or services and on the interaction between these. A number of extensions of the WSF can be identified [2,3]. On the structural level, composing services is not part of the WSF. The description of services is limited to syntax and type aspects. On the descriptional level, no support is provided for functional and non-functional semantical properties. Whereas the first development phase of the WSF has focussed on infrastructure, other aspects have become more important since then. Besides infrastructure support for service deployment, the development of service-based software systems is now the focal aspect.

The combination of the WSF core with the Semantic Web [4], in particular ontology technology [5], can provide an essential step forward that introduces meaning to Web services and that provides the foundations to enable a software component-style composition of services to service processes [6].

Previous work on the combination of the Semantic Web and Web Services has often focussed on modelling and language aspects [2,3,7]. More service architecture-oriented treatments have neglected the semantical aspects [1]. Here, our aim is to identify the common aspects of semantic Web services approaches [8,9] and capture these in a conceptual architecture. We identify models, languages, infrastructure, and stakeholder activities as the central layers of this architecture. We focus on technologies, such as ontology technology, that can play an integrating role in this endeavour. These technologies are central building blocks of the architecture.

Such a conceptual architecture can form the underlying foundation of a methodology for semantic Web services development and deployment. It provides a taxonomy for a development and deployment platform. A major aim of our proposed architecture is to link models, languages, infrastructure, and activities. The conceptual architecture results from an empirical analysis of work on semantic Web services and Web services infrastructures such as [2,3,9,10,11,12,13,14,15].

In Section 2, we introduce the Web services framework and semantic services. We present our conceptual architecture and the research that led to this architecture in Section 3. In Section 4, we investigate model and language aspects of the conceptual architecture. Infrastructure and activity aspects of the architecture are subject of Section 5. We end with some conclusions.

2 Semantic Web Services – Background

Before we introduce our conceptual architecture, we give an overview of the background technologies – Web services, ontologies, and semantic Web services – involved.

2.1 Web Services

A *Web service* is defined as a provided software system identified by a URI, whose public interfaces are defined and described using XML [1]. Other software systems, i.e. requesters of the service, may interact with the provided Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols. A wider scope of the service notion includes distributed object – or Web-mediated – services.

In the Web services framework WSF [1], a description language, WSDL (Web Services Description Language), is used to describe syntax and type aspects of services, in particular message formats and message exchange details, and their binding to a communications protocol. A registry service, UDDI (Universal Description, Discovery, and Integration), is a repository that allows providers to publish service descriptions and service requesters to search for services. A protocol, SOAP (Simple Object Access Protocol), is used to invoke services.

2.2 Ontology Technology

The Semantic Web initiative aims at making the Web more meaningful and open to manipulation by software applications [4]. A logic-based approach based on knowledge representation and reasoning forms the backbone. Annotations to Web resources express meaning, which can be used by software agents to extract semantical information about the resource. The requirement for this to work is a precise, shared understanding of these annotations.

Ontologies provide a solution for this requirement. Ontologies define terminologies and semantical properties. Essentially, ontologies are hierarchical definitions of concepts of a domain and descriptions of the properties of these concepts. Logics such as description logics [5] provide the reasoning support. Integrated into an ontological Web framework based on OWL – the Web Ontology Language – sharing of ontologies becomes possible.

Knowledge representation through ontologies can be utilised to describe Web services. Two types of knowledge relevant to the services context need to be represented ontologically. *Domain knowledge* captures entities from the application domain and their properties – domain modelling is a widely accepted requirements engineering method. *Software knowledge* captures software artefacts and their properties. Expressing semantics and reasoning about it is the central goal. Software knowledge is often expressed by incorporating domain knowledge. Description and reasoning facilities provided by ontologies are essential building blocks of a semantic Web services approach.

2.3 Semantic Web Services

Different *development scenarios* for services-based software systems involving requester and provider can be imagined (Fig. 1): collaborative development of services or provider-based development of services with human or automated discovery. In any case, the existence of requester and provider makes sharing of knowledge about services and their context through ontologies necessary. Often, the automation of development and deployment processes involving Web services – from the discovery to the final invocation – is seen as the ultimate goal [3]. The degree of automation determines the scalability of the architecture. A cornerstone of such an endeavour is the support of semantics [2,3,7,12,13,14,15,16]. The WSF focuses on message format and message exchange mechanisms to provide and invoke services. In addition, various semantical properties of services are relevant for a service user, e.g. [2,10]:

- *Transitions*: the abstract behaviour is often represented in a transitional form describing in/out-transitions.
- *Dependencies*: the interaction of a requestor with a service might be constrained, i.e. operations can only be invoked following an *interaction pattern* or *protocol*.

- *Interaction processes*: the internal process and interaction structure, possibly involving other services, that provide the functionality for the service. While similar to the external interaction patterns, this needs to address *data and control flow* and *synchronisation* more explicitly.

In order to support semantic service reuse and composition, ontology technology has been proposed as a means based on successful techniques used in WSF extensions – domain modelling [3], design-by-contract [2], and process composition [17]. We propose to integrate these into a coherent conceptual architecture for semantic Web services. Semantical properties – including behaviour and dependencies – enable the reuse of services and their independent composition, resulting in a platform-specific development and deployment style for Web services. Ontologies can represent semantics in a shared, machine-processable format.

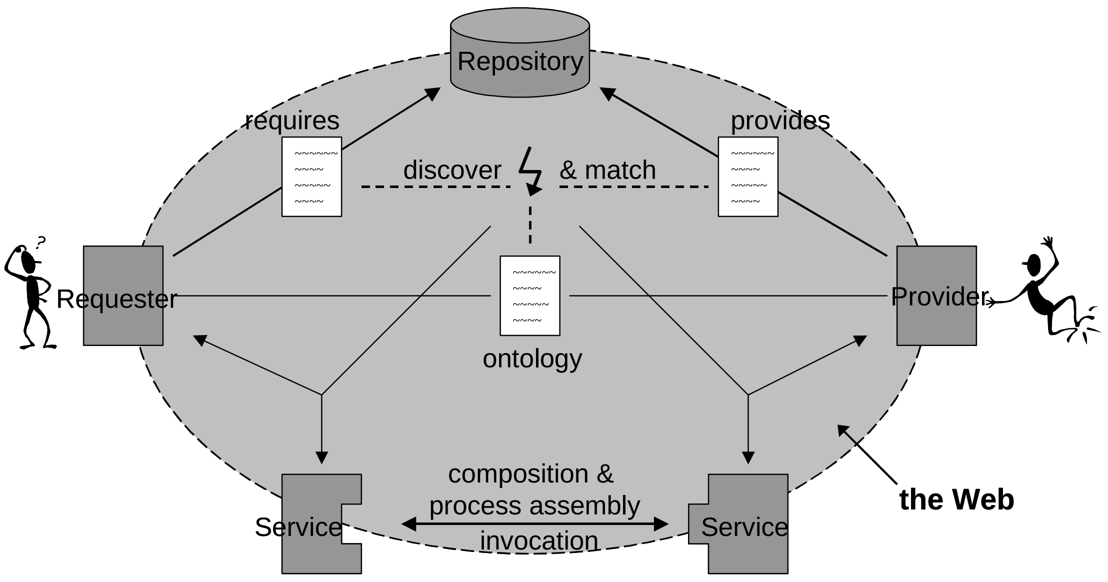

Figure 1. Semantic Web Services Development and Deployment.

3 A Conceptual Architecture for Semantic Web Services

The conceptual architecture that we have developed shall be introduced in this section. The architectural is based on an empirical analysis of current research in this area that we have carried out to identify commonly addressed issues and also areas where still work has to be done.

3.1 Development and Deployment of Service-based Software Systems – some Observations

Services for the WSF are coherent collections of operations described in an interface and provided to a user. Often a service is seen as an abstract notion that must be implemented by a concrete agent [1]. There are consequently two aspects of services:

- *Internal View*. Services provide functionality through operations. These operations might encapsulate an internal state; their behaviour needs to be coherent in terms of the service they provide.
- *External View*. Two different roles – those of requesters and providers – are immediately apparent, see Fig. 1. The interaction between these – either humans or software agents acting on their behalf – is a central aspect. For instance, agreement on the service semantics and the mechanisms of message exchange are vital.

Both the internal and the external view need to be looked at in the context of service development and deployment:

- *Models and Languages*. These provide the foundations necessary to model services as coherent sets of operations. All service aspects relevant for a potential user need to be captured in abstract descriptions. In open environments, representing and sharing knowledge is central.
- *Infrastructure and Activities*. Specific interactions are required between requester and provider – activities such as discovery, composition, and invocation of services. These have to be supported by an adequate infrastructure consisting of protocols and tools.

Service-based platforms such as the WSF are based on remote procedure call mechanisms, adding a publication and discovery infrastructure, see Fig. 1. We suggest an extension of these platforms towards a service-oriented development and deployment architecture by adding further development infrastructure, e.g. semantics and composition. We will introduce our conceptual architecture, which provides an abstract model of the development and deployment context, in Section 3.2. This architecture integrates the various aspects involved, including underlying conceptual models, languages, development and deployment infrastructure, and activities of the stakeholders.

3.2 The Conceptual Architecture Definition

In the first-generation WSF, there is no support for semantical descriptions or the composition of services. For instance, business processes cannot be modelled. Some attempts have already been made to rectify this. Web service composition languages such as WS-BPEL allow Web services to be composed through choreography and orchestration [18,19]; OWL-S (formerly known as DAML-S) [2] is a Semantic Web-compliant ontology for Web service description, and the Web Service Modelling Ontology WSMO [16], which is based on the Web Service Modelling Framework WSMF [3], is another ontology-based modelling approach.

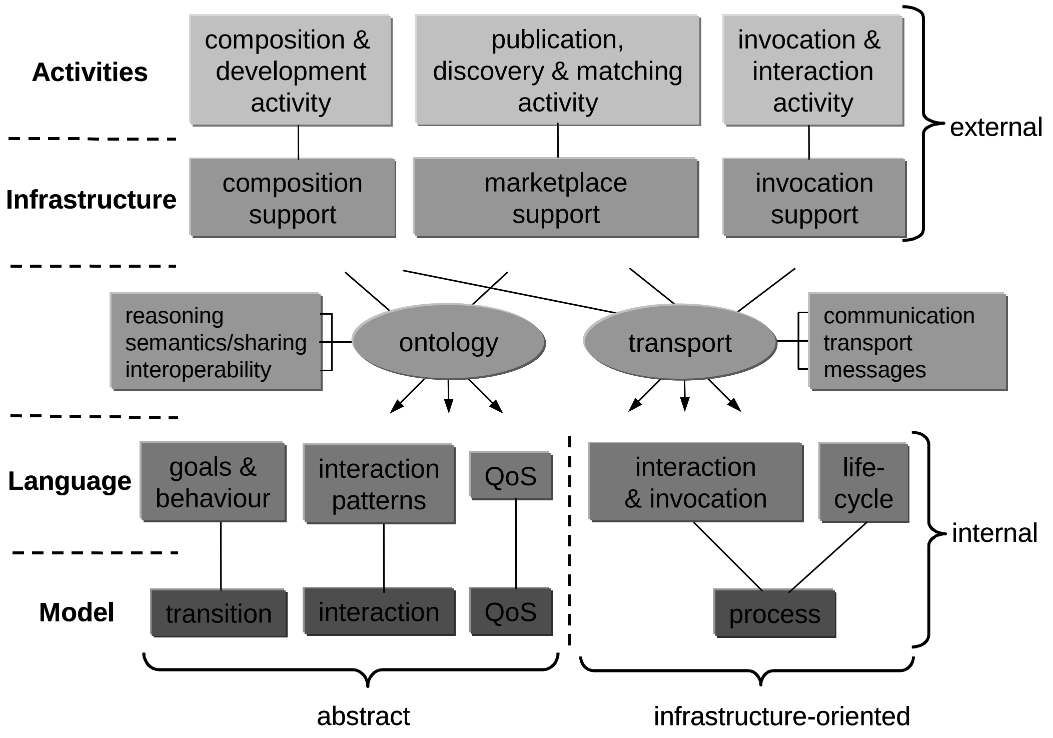

Figure 2. A Layered Conceptual Architecture for Semantic Web Services.

For our conceptual architecture, we propose to follow the route taken by OWL-S and WSMO/WSMF towards an extended WSF and base our architecture on *ontology technology* as a central element. Moreover, we add a new aspect. Component technology [6] aims at modular composition of software systems from self-contained, reusable components described by contract-based interfaces and explicit context dependencies. The *principle of composition* in the WSF is *process assembly*. Looking at component technology explains our motivation. WSDL descriptions do not make dependencies on other services explicit; they do not state their infrastructure requirements – which would, however, be a prerequisite for reuse and independent composition. The Web services platform focuses on messages, i.e. sees the description of message formats and their exchange at the core, rather than the effects that are caused by message exchange. We focus on service semantics in the context of service composition through choreography and orchestration where dependencies have to be made explicit.

The *service development and deployment aspects* (model, language, infrastructure, and activity) form the different *layers* of our *conceptual architecture* – see Fig. 2. We will discuss the central architecture aspects – model and language, infrastructure and activities – in Sections 4 and 5, respectively.

Fig. 2 indicates that the link layer between internal and external layers is an essential component of the architecture. The two lower layers cover aspects that have already been addressed by the community. The upper two layers are based on the philosophy of the WSF as a software engineering platform. Consequently, the two link components of the architecture aim to integrate the layers:

- The *ontology component* provides semantics and a notation for models and languages and is the basis for infrastructure elements and activities. The Web standards XML, RDF, and OWL form the platform for interoperability, semantics, and reasoning. This component is more development oriented. We will focus in the remainder on the ontology component.
- The *transport component* provides distribution technology. It is based on the core protocols for the Web services platform, consisting of the Internet protocols, Web-specific protocols such as HTTP and XML Protocol, and Web service-specific choreography and orchestration standards. The transport component addresses the technical side of composition and acts as support infrastructure for other activities. This component is more deployment oriented.

The WSF as the deployment platform determines to some extent the development part of this architecture.

3.3 Towards a Semantic Web Services Development and Deployment Methodology

The conceptual architecture provides an architecture (in a technical sense) for a *semantic services development and deployment methodology*, identifying the overall task of the methodology, its core components, and how these work together. Automation of stakeholder activities in a shared and distributed environment, such as the discovery and selection of suitable services for a requester, requires a new distributed type of development and deployment methodology in the Web services environment based on joint activities, shared knowledge and artefacts, and reuse – supported by a distributed infrastructure geared towards this purpose.

A methodology for semantic Web services development and deployment is needed to establish the WSF as a software platform. A software engineering methodology consists of a collection of methods that address activities to solve particular problems. Principles and theories determine these methods. Models and languages based on the principles and theories together with infrastructure support enable these (usually activity-specific) methods to be applied.

Our aim is to provide an architecture for this methodology by identifying the major tasks of a software engineering approach for this context (see Fig. 1), by identifying the central components of this methodology, and to capture how these relate to each other. Defining and validating these methods based on this architecture would be the next step towards a methodology.

The lower two layers of the conceptual architecture (see Fig. 2) are based on existing research, e.g. [2,3,7,8,9,10], showing success in this direction in terms of models and languages. The upper two layers are induced by the philosophy of the WSF, as indicated in Fig. 1, and core standards and extensions as they are defined by the W3C and other organisations. For a methodology to be established, these layers need to be linked and integrated. We have identified ontology and transport technologies are suitable links for the development and deployment aspects, respectively.

3.4 An Illustration of the Conceptual Architecture

In Fig. 3, we have illustrated a *sample service description* – representing a provided *online bank account* service. The service description lists a number of individual operations. We have used pseudocode for signatures and pre-/postconditions based on the ontology languages – a formulation in proper description logic, a foundation for ontology languages, will be discussed later on. We have limited the specification in terms of pre- and postconditions to two operations. The specification here is not meant to be complete; it aims to illustrate the architecture concepts.

In the context of the WSF, a potential requester of a service can use a similar specification as a requirements definition and search a repository or marketplace to discover provided services that match. The requirements specification forms a query. The ontology language is the query language. The ontology provides the vocabulary for the query. A query should result ideally in the identification of a suitable (i.e. matching) description of a provided service.

```
Service BankAccount
   Signatures and Pre-/Postconditions
      login
         inSign   no:int, user:string
         outSign  void
      balance
         inSign   no:int
         outSign  real
      lodgement
         inSign   no:int, sum:real
         outSign  void
         preCond  true
         postCond balance(no) = balance(no)@pre + sum
      transfer
         inSign   no:int, dest:int, sum:real
         outSign  void
         preCond  balance(no) >= sum
         postCond balance(no) = balance(no)@pre - sum
      logout
         inSign   no:int
         outSign  void
   Service Process
      login;!(balance+lodgement+transfer);logout
```

Figure 3. A Provided Online Bank Account Service.

In the following sections, we will explain and illustrate the conceptual architecture using the banking example introduced here. We will use concrete technologies to demonstrate how the frame defined by the architecture can be filled. Section 4 will look at the two lower layers (models and languages); Section 5 will focus on the upper two layers (infrastructure and activities).

4 Models and Languages

The semantic description of a service in a shared knowledge representation format, based on common domain and computation models, is a central element of our conceptual architecture for services. These models need to be made available through appropriate languages. Knowledge engineering becomes therefore a pivotal technology.

4.1 Service Models

Taking the concepts of the various semantic Web services approaches on board, we can identify essential aspects of Web services that should form core elements of Web services modelling and specification:

- *external descriptions* of the service in terms of its *goal* or *purpose* (assumptions and characterisation of the expected outcomes in terms of domain concepts), the *effect* (how acceptable input is transformed into output), and interaction protocols (ordering of operations) – these aspects often form the *contractual information*,
- *internal descriptions* of composed services including data and control flow that coordinates interactions between subservices,
- *interaction infrastructure descriptions* for services consisting of input/output data formats and ports and the protocol binding to handle the message exchange.

We have captured these aspects through the models and languages of our conceptual architecture.

In particular computational aspects of service properties need to be based on appropriate models that underlie semantical description and reasoning. We have outlined the types of information needed to adequately represent service behaviour, including input/output behaviour, interaction protocols, and service composition and communication [20]. Three types of computational models can address these aspects:

- *Transitional model*: an abstract view on services and in particular on service operations is the transitional input-output behaviour. These descriptions are often called contractual information; pre- and postcondition-based techniques are usually used [21]. A suitable model that covers the contractual aspects of the service is a state-transition model defining operations as transitions in a state space.
- *Interaction model*: an abstract view on a service's interactions with a service user. Often, only certain interaction patterns based on the offered operations are possible. Constructors such as sequence, choice, and iteration can be used to formulate these interaction protocols. Again, a state-transition model, here with

constructors to compose transitions, is suitable to model the interaction behaviour of a single service or operation and to capture the service interaction patterns.

- *Process model*: a more detailed view on interactions between services, viewing services as interacting processes. The interaction between a service and its user and also between the internal subservices used to provide the overall service needs to be addressed [11]. In both cases, the focus is on sending and receiving messages, and on the synchronisation between processes. A classical process model, as formulated in process algebras, can form the basis here to cover process synchronisation aspects for service invocations.

Quality-of-Service models complement the range of models [15] – which can range from software attributes such as maintainability, security, and efficiency to aspects such as pricing. We will ignore quality-of-service attributes here and focus on functional behaviour.

We can classify the conceptual models (and the corresponding languages) into two categories: *abstract* and *infrastructure-based*. Only the process model falls into the latter category since it refers to an abstraction of the WSF infrastructure; the other models refer to abstract, infrastructure-unrelated service properties.

4.2 Abstract Service Description

We suggest an ontology language [2,3] to introduce a description notation for abstract Web service properties. We use a description logic here [5], which underlies an ontology language such as OWL. Description logics are based on the idea of defining a concept in terms of its properties in relation to other concepts. The language we use here is an extension of classical description logics to address the semantical aspects identified in Section 2.3.

Describing Services as Processes.

Central to the modelling and composition aspect of the conceptual architecture is to understand services as processes. Service processes and service-oriented composition have not been addressed in the current WSF. A process view – which we can capture in ontological terms – allows us to include process composition and interaction. Moreover, it helps us to formalise (and eventually automate) stakeholder activities. Consequently, ontologies describing service properties in their domain context should support service composition and processes.

Description logic [5] knows two basic elements. *Concepts* are classes of objects with the same properties. For instance, in a banking application, an *Account* is a central concept. *Roles* are relations between concepts. Roles express properties of concepts in relation to other concepts. An *Account* can be characterised by a *balance*-property, which relates *Account* with a *Numerical* value concept.

Concept descriptions are constraints based on simple set-theoretic operators and quantified expressions. *Operators* include $\neg$, $\cup$, $\cap$, and $\rightarrow$ with their usual set-theoretic meaning. For instance, $\mathit{CurrentAccount} \cup \mathit{SavingsAcconut}$ is the concept that describes the union of both account classes. The *value restriction* $\forall R.C$ for a given concept *C* restricts the values of role *R* (as a relation) to elements that satisfy *C*; the *existential quantification* $\exists R.C$ requires the existence of a role value satisfying *C*. For instance, an *Account* could be characterised by a numerical balance property $\exists balance.Numerical$.

Different ways to model services have been suggested. In [2,13], services are represented as concepts, with properties associated through roles. In [10,11], services are modelled as roles, interpreted by accessibility relations between states. Essential for our conceptual architecture is to provide operators to compose services based on the idea of services as interacting processes.

Goals and Behaviour.

We can associate pre-state and post-state descriptions with services and their operations. Properties of these states (possibly in different formats) can be expressed using roles.

- *Goals* are abstract specifications of service behaviour [3]. *Assumptions* are pre-state properties that summarise domain concept definitions relevant to the service, such as 'account' or 'balance'. The goal itself is an expression of the expected outcome of a service execution, usually involving the assumed concepts. A banking example is the expectation that after lodging money into an account, the balance will have increased.
- *Contractual information* about behaviour can be specified in terms of pre- and postconditions [2]. These conditions are expressions relating to parameters of the service operation signature, possibly involving domain concepts. For a lodgement service, the sum transferred into an account plus the pre-state balance yields the post-state balance. Contracts are refinements of goals [3].

Often, extensions of classical ontology languages are necessary to enable goal and in particular contractual specifications. In [10,11], it is necessary to introduce names into role expressions in order to express parameters. An example of a postcondition specification is

$$\forall\ lodgement \circ Sum_N\ ;\ postCond.\ equal(Bal;\ Bal@pre + Sum)$$

saying that a transitional role *lodgement* (which is a service operation) is applied to parameter name Sum_N , and that after execution the balance *Bal* is increased by *Sum* in the post-state (which is the postcondition).

Interaction Protocols.

Interaction protocols are pattern expressions constraining the order in which operations of a service can be invoked. In order to facilitate these expressions, we need introduce control flow operators – e.g. ; (sequence), + (choice), and ! (iteration) – to support the *interaction model* [11]. For instance,

$$login;\ !(lodgement + transfer + balance);\ logout$$

expresses that after logging in to the online banking system, money can be repeatedly lodged or transferred or the balance can be requested, before the user logs out.

4.3 Infrastructure-based Service Descriptions

Service Synchronisation and Invocation.
In order to deal with synchronisation and actual interactions described in the *process model*, we need to take another view on service operations. So far only considered as transitions in state-based systems, we need to consider both the requester and the provider of these transitions. For instance, an automation of accesses to UDDI repositories would require such a process communication view. A description notation can build up on the ontology language for abstract service description by adding process calculi elements.

- *Ports*. The service operation names define ports that, if synchronised with another port from another service process, can form an interaction channel.
- *Orientation*. Each port carries additional information indicating whether it is used for sending or receiving on the channel. We use $op(a)$ for input (receiving) and $op\langle a\rangle$ for output (sending) following [17] instead of an abstract role expression such as $op \circ a$ that we have introduced in Section 4.2.

The service process expression

$$getBalance(bal);\ setBalance\langle bal + ldg\rangle$$

based on the abstract expression

$$getBalance \circ bal_N\,;\ setBalance \circ (bal_N + ldg_N)$$

expresses that the specified service receives input *bal* using port *getBalance* and then sends the value of *bal* + *ldg* back to the remote service using port *setBalance*.

Service Lifecycle.
A notation to express service process interaction can be used to formalise an *activity-based lifecycle view on services* [22] – which leads us into the infrastructure and activity aspects of our conceptual architecture. Addressing the complete software lifecycle is an essential aspect of software engineering methodologies. A service lifecycle is determined by activities such as matching, composition, and execution, and supported by infrastructure facilities such as repositories, brokers, and protocols. The lifecycle can be expressed as a process where different ports represent the infrastructure to support activities. A service port actually facilitates several activities:

- *Contract*. Using contract ports, matching constraints guard the establishment of an invocation infrastructure using different types of invocation ports.

- *Invocation* and *Reply*. Invocation ports allow a service to be invoked and necessary parameters to be passed. Message type aspects constrain this interaction. Often, a service reply is communicated on another channel.

A provider lifecycle based on these service port types could follow the pattern

$$serv_{CTR}(serv_{INV});\ !(\ serv_{INV}\ (a,\ serv_{REP});\ serv_{REP}\langle f(a)\rangle\ )$$

with the activities contract matching $serv_{CTR}(serv_{INV})$, invocation $serv_{INV}\ (a,\ serv_{REP})$, and replying $serv_{REP}\langle f(a)\rangle$ for the contract, invocation, and reply ports $serv_{CTR}$, $serv_{INV}$, and $serv_{REP}$, respectively [10]. The interaction pattern expresses that, after a contract match, a service can be invoked and a reply can occur repeatedly. This would formalise the UDDI-supported matching of WSDL descriptions of Web services and their invocation using SOAP in the WSF [1].

5 Infrastructure and Activities

The core task of a services platform is to facilitate service invocation, but it also needs to support other stakeholder activities such as composition or publication and discovery, see Fig. 2. The activities are core elements of a development and deployment methodology. The basic requirements for our conceptual architecture arise already from the infrastructure required for discovery and invocation in the WSF. Semantic description and composition services can be layered on top [23].

5.1 Infrastructure Tools and Facilities

Infrastructure tools and facilities aim to support the development and deployment activities. They build up on *distribution technology*, i.e. the layered transport model, and *knowledge and semantics technology*, i.e. the layered ontology model; see Fig. 2. Central infrastructure components are:

- *Marketplace* – based on transport and ontology technology – to support publication, discovery, and matching based on semantics-enabled UDDI and WSDL.
- *Composition* based on transport and ontology technology – to support composition through orchestration and choreography based on service semantics and interaction.
- *Invocation* – based on transport technology – to support interaction for service invocation based on Web protocols.

Infrastructure tools and facilities such as repositories, brokers, composition engines, and protocols are usually provided through suitable APIs.

The central activities invocation and execution of Web services shall be based on the *layered transport model*. Starting at the bottom, message types characterise the payload of *messages*. *Transport bindings*, e.g. SOAP, define the message layout. Exchange-related aspects – protocol properties such as resending rules – are also part of the transport model. The essential elements are the *interaction processes* – defining the sequencing of send and receive operations.

5.2 Development and Deployment Activities

A Web services platform needs to enable stakeholders (providers and requesters) to carry out development and deployment activities. Building up on core technologies (transport/distribution and ontologies), activities such as discovery, composition, and invocation and interaction need to be facilitated. Distribution is a property of a Web services architecture. Both development and deployment activities take place in this distributed context. A simplified distributed development and deployment process based on discovery, matching and invocation/interaction activities can be modelled through a lifecycle protocol; see Section 4.3.

Publication, Discovery, and Matching.
Requesters need to find and compare service providers for the services they need. The infrastructure that the WSF provides is the UDDI registry. Providers can publish descriptions of their services in these registries which can then be searched.

The central difficulty is *matching* [24], i.e. to find the service(s) that most closely match the requirements of the requester. In an automated setting, a software agent will use requirements formulated by the requester in a shared ontology language to search repositories for matching services. A notion of matching needs to capture the idea of satisfaction or refinement. A provided service needs to be at least as good as the requested one. In an ontology language, the *subsumption* concept – the subclass relationship between concepts or roles – captures this. A service matching notion needs to be composite, as services themselves and also their descriptions are composite. For each of the individual aspects we need some kind of metric to decide matching. Each of them is supported by an underlying conceptual model (Section 4.1).

- *Goals and contractual information* – based on a transitional model. For instance, refinement-based notions of matching can be used; weakening the precondition and strengthening the postcondition is a standard choice [21,24].
- *Interaction protocols* – based on an interaction model. A notion of simulation can form the basis of matching [17].
- *Processes* – based on a process model. A notion of simulation can again form the basis of matching here.

In all cases, the matching constructs can imply subsumption and can therefore be integrated into an ontological framework, see [10,11]. Subsumption allows us to capture widely used software development concepts such as refinement and simulation. Service-based matching can be deployed if a provider service is to be integrated into an existing process. For instance, a provided lodgement service $lodgement \circ Sum_N$ characterised by the postcondition *equal*(*Bal,Bal@pre+Sum*) *and logged(lodgement)* satisfies (or refines) the requirement *equal*(*Bal; Bal@pre* + *Sum*).

Description and reasoning using ontology technology is the central contributor to discovery and matching activities. Reasoning can be facilitated through the use of description logic-based inference tools [5] or through the use of transition system and automata-based approaches for verification [25].

Composition and Development.
Composition can be both a development-time and a run-time activity. Services can be composed to composite service processes. An expression like *login*; !(*lodgement + transfer + balance*)*; logout* describes a composite service process. This process might be assembled from service different providers. In that case, each individual service has to fit into the context by matching the process requirements – the process is here the client of the provided services.

We can distinguish client-side and provider-side composition of provider services, and provider-side constraining of provider services followed by client-side composition [6]. The variants can be characterised by the degree of cooperation and the degree of automation that is enabled. Automation is important for run-time composition.

Invocation and Interaction.
In an automated approach, activities of the provider and the requester have to be synchronised. We can define inference rules based on the ontology language that govern these synchronisations at runtime [22]. Important is here that different communication channels are used for retrieval and matching on the one hand and later service invocation interactions on the other – as expressed in the service lifecycle example. An inference rule, such as the following connector rule

$$\frac{\forall\ servA_{CTR}(servA_{INV})\ .\ postCond_{A}\quad and\quad \forall\ servB_{CTR}(servB_{INV}).\ postCond_{B}}{\forall\ servA_{CTR}(servA_{INV})|\ servB_{CTR}(servB_{INV}).\ postCond_{A} \cap postCond_{B}}$$

define and constrain the execution behaviour. The rule describes the establishment of an invocation connection *INV* between two services using contract channels *CTR*. The two services in this situation could be a client-side process *servA* and a server-side process *servB*, which might be executed in parallel and which might interact.

Internet protocols provide the basic transport infrastructure. On top of these, service-specific protocols such as SOAP provide an RPC mechanism. Ontology-based interaction patterns describe the interaction behaviour of services. The inference techniques need to be implemented based on the existing interaction infrastructure. Engines for SOAP-based service invocation and interaction can be extended to deal with semantical checks.

6 Conclusions

Semantic Web services are an increasingly important topic. However, a coherent conceptual architecture for semantic Web services that captures and integrates recent developments is currently lacking. We have addressed the integration of the different aspects models, language, infrastructure, and activities through our conceptual architecture. Several directions – including ontological modelling and process composition – are currently investigated. We have integrated these two central aspects into our conceptual architecture.

The proposed conceptual architecture is the result of an empirical investigation into various approaches in the Web services context. It aims to act as a taxonomy and through its linkage of models, languages, infrastructure, and activities; it helps us to better understand the problems of Web services development and deployment. It aims to provide a foundation for a semantic Web service-oriented development methodology. It captures current developments such as the Web service framework WSF, OWL-S, WS-BPEL, and others, and places these in the wider development context. We have demonstrated the suitability of the central architecture aspects and how the architectural framework can be realised using concrete technologies illustrated by an online banking example.

A high degree of automation is a requirement for the future of the Web services framework – scalability and, therefore, the success of the framework depend on it. Automation requires shared semantics in a distributed, heterogeneous environment for development and deployment. Ontology technology is a solution to this problem. (The technical aspects of the ontology framework we have presented here are only indicative of what is needed on the language and model side.) Ontologies are reflected in all facets of our conceptual architecture – models, languages, infrastructure, and activities. Ontologies can capture the process-oriented view on services and can provide the necessary features to support the corresponding activities. One of the aspects that we neglected in our discussion are quality-of-service issues. They include various aspects including performance and security. Despite the importance of these aspects, we have restricted our focus here on functional behaviour.

In addition to semantics and ontologies, services as processes is a notion that is central to enhance the Web services framework – and that needs to be made explicit in conceptual architectures and service infrastructures supporting the framework. Service choreography and orchestration are two terms that capture the idea of business and workflow process definition based on service process composition. We have demonstrated that ontology technology can be used to capture services as processes and also process composition.

Our analysis of a number of semantic Web services extensions and process composition approaches indicates progress towards a new methodology for composition-based semantic service development and deployment – to be supported by a generic conceptual and architectural framework. The Web environment requires suitable workflow processes in particular for service development. Our proposed services-oriented development and deployment architecture is different in many ways from the current Web services framework WSF. It exhibits characteristics of a component framework. It supports a different style of development and deployment embracing composition and workflow processes. It creates a space for composable, Web service-enabled components. Our achievement is the introduction of an architecture for these service components that connects the aspects model, language, infrastructure, and activity based on coherent Web-based ontology and transport technologies.

References

1. World Wide Web Consortium. *Web Services Framework*. http://www.w3.org/2002/ws, 2003.

2. DAML-S Coalition. DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342. pp. 279–291. Springer-Verlag, 2002.

3. D. Fensel and C. Bussler. The Web Services Modeling Framework WSMF. *Electronic Commerce Research and Applications*. 2002.

4. W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.

5. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

6. C. Szyperski. *Component Software: Beyond Object-Oriented Programming – 2nd Ed.* Addison-Wesley, 2002.

7. S. McIlraith, T.C. Son, and H. Zheng. Semantic Web Services. *IEEE Intelligent Systems* 16(2). March/April. 2001.

8. C. Pahl. A Conceptual Framework for Semantic Web Services Development and Deployment. In L.-J. Zhang and M. Jeckle, editors: *European Conference on Web Services ECOWS 2004*. Springer-Verlag. LNCS 3250. pp. 270-284. 2004.

9. E. Motta, J. Dominigue, L. Cabral, and M. Gaspari. IRSII: A Framework and Infrastructure for Semantic Web Services. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*. pp. 306–318. Springer-Verlag, LNCS 2870, 2003.

10. C. Pahl. An Ontology for Software Component Matching. In *Proc. Fundamental Approaches to Software Engineering FASE'2003*. Springer-Verlag, LNCS Series, 2003.

11. C. Pahl and M. Casey. Ontology Support for Web Service Processes. In *Proc. European Software Engineering Conference and Foundations of Software Engineering ESEC/FSE'03*. ACM Press, 2003.

12. R. Zhang, I.B. Arpinar, and B. Aleman-Meza. Automatic Composition of Semantic Web Services. In *Proc. International Conference in Web Services ICWS'2003*. 2003.

13. S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. World-Wide Web Conference WWW'2002*. 2002.

14. L. Chen, N. Shadbolt, C.A. Goble, F. Tao, S.J. Cox, C. Puleston, and P.R. Smart. Towards a Knowledge-Based Approach to Semantic Service Composition. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*. Springer-Verlag, LNCS 2870, 2003.

15. C. Zhou, L. Chia, and B. Lee. DAML-QoS Ontology for Web Services. In *International Conference on Web Services ICWS 2004*. IEEE Press. pp. 472-479. 2004.

16. R. Lara, D. Roman, A. Polleres, and D. Fensel. A Conceptual Comparison of WSMO and OWL-S. In L.-J. Zhang and M. Jeckle, editors: *European Conference on Web Services ECOWS 2004*. Springer-Verlag. LNCS 3250. pp. 254-269. 2004.

17. D. Sangiorgi and D. Walker. *The π-calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.

18. N. Desai and M. Singh. Protocol-Based Business Process Modeling and Enactment. In *International Conference on Web Services ICWS 2004*. IEEE Press. pp. 124-133. 2004.

19. J. Rao, P. Küngas, and M. Matskin. Logic-Based Web Services Composition: From Service Description to Process Model. In *International Conference on Web Services ICWS 2004*. IEEE Press. pp. 446-453. 2004.

20. F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, 28(11):1056–1075, 2002.

21. B. Meyer. Applying Design by Contract. *Computer*. pp. 40–51, October 1992.

22. C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. *ICALP'2002 Workshop on Formal Methods and Component Interaction*. Elsevier Electronic Notes on Computer Science ENTCS, Vol. 66, No. 4. July 2002.

23. S. Dustdar and W. Schreiner. A Survey of Web Services Composition. *International Journal of Web and Grid Services*, 1(1), 2005.

24. A. Moorman Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.

25. D. Kozen and J. Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B.* pp. 789–840. Elsevier, 1990.