A Conceptual Framework for Semantic Web Services Development and Deployment

Claus Pahl

Dublin City University, School of Computing
Dublin 9, Ireland
Claus.Pahl@dcu.ie

Abstract. Several extensions of the Web Services Framework have been proposed. The combination with Semantic Web technologies introduces a notion of semantics, which can enhance scalability through automation of service development and deployment. Ontology technology – the core of the Semantic Web – can be the central building block of this endeavour. We present a conceptual framework for ontology-based Web service development and deployment. We show how ontologies can integrate models, languages, infrastructure, and activities within this framework to support reuse and composition of semantic Web services.

Keywords: Web Services, Semantic Web, ontology technology, conceptual development and deployment framework.

1 Introduction

Opening the Web for software applications is the objective of the Web Services Framework WSF [1]. Services are self-contained computational entities, used as is by service requesters, and made available through the infrastructure provided by a provider. The focus is on the boundaries of systems and on the interaction between systems.

A number of shortcomings of the Web Services Framework WSF can be identified [2–4, 1]. On the structural level, composing services is not part of the WSF. The description of services is limited to syntactical and type aspects; no support is provided for functional and non-functional semantical properties. The combination with the Semantic Web [5], in particular ontology technology [6, 7], 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 [8, 9].

Previous work on the combination of the Semantic Web and Web Services has often focussed on modelling and language aspects [2, 3]. More architecture-oriented treatments have neglected the semantical aspects [1]. Here, our aim is to identify the central aspects of a conceptual framework for semantic Web services architectures [10]. We address models, languages, infrastructure, and stakeholder activities – and illustrate the integrating role that ontology technology can play

in this endeavour. Such a framework can form the underlying foundation of a methodology for semantic Web services development and deployment. It provides a taxonomy for a Web services development and deployment platform. A major aim of the framework is to link models, languages, infrastructure, and activities. The conceptual framework results from an analysis of our own language-oriented work [11, 12], but also related work on semantic Web services and Web services infrastructures such as [13, 14, 2, 15–17, 3, 18–20, 4, 21–23, 10]. We aim to capture these in a generic conceptual framework.

In our framework, particular models for services are essential and need to be prescribed. This is a clear deviation from the WSF focus on interfaces and interactions. Our notion of a Web services architecture is connected to a different style of service development and deployment than anticipated by the WSF – based on principles such as composition and reuse and a notion of services as processes.

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, sharing knowledge and artefacts, and reuse – supported by a distributed architecture geared towards this purpose.

In Section 2, we outline the basics of Web services and introduce the rationale behind our conceptual framework. In Section 3, we investigate model and language aspects of a conceptual framework for Web services. Infrastructure and activity aspects of the framework are subject of Section 4. We end with some conclusions.

2 Web Services

Our objective is to introduce a conceptual Web services framework supporting semantic service reuse and composition. We propose ontology technology as a means to integrate successful techniques used in WSF extensions – domain modelling [3] or design-by-contract [2] – into a coherent framework.

2.1 A Conceptual Framework for Service-oriented Architectures

The Web Services Architecture WSA [1] defines "a Web service [as] a software system identified by a URI, whose public interfaces are defined and described using XML. Other systems may interact with the 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, i.e. software agents providing the service functionality that are not necessarily part of the Web environment themselves[1].

Services in service-oriented architectures (SOA) are coherent collections of operations described in an interface and provided to a user. Often a service is

[1] This would allow us to see the WSF as a meta- or interoperability framework between middleware platforms.

seen as an abstract notion that must be implemented by a concrete agent [1]. There are consequently two aspects of services:

- The *internal view*. Services provide functionality through operations. These operations might encapsulate an internal state; their behaviour certainly needs to be coherent in terms of the service they provide.
- The *external view*. Two different roles – those of requesters and providers – are immediately apparent. 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 external view need to be looked at in the context of service development and deployment. A *conceptual framework* provides an abstract model of the development and deployment context that integrates the various aspects, including underlying conceptual models, languages, development and deployment infrastructure, and activities of the stakeholders involved:

- *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 public environments, representing and sharing knowledge is central.
- *Infrastructure and Activities*. Specific interactions are required between requester and provider – activities such as discovery of services, composition, and invocation of service agents. These have to be supported by an adequate infrastructure consisting of protocols and tools.

The *service development and deployment aspects* (model, language, infrastructure, and activity) form the different *layers* of our *conceptual framework* – see Fig. 1.

Service-oriented architectures (SOA) are based on remote procedure calls RPC, adding a publication and discovery infrastructure. Our framework suggests an extension of SOAs towards a service-oriented development and deployment architecture by adding further development infrastructure, e.g. semantics and composition.

2.2 Semantic Web Services

Different development scenarios for services involving requester and provider can be imagined: 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 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 in this context determines the scalability of the framework. A cornerstone of such an endeavour is the support of semantics for

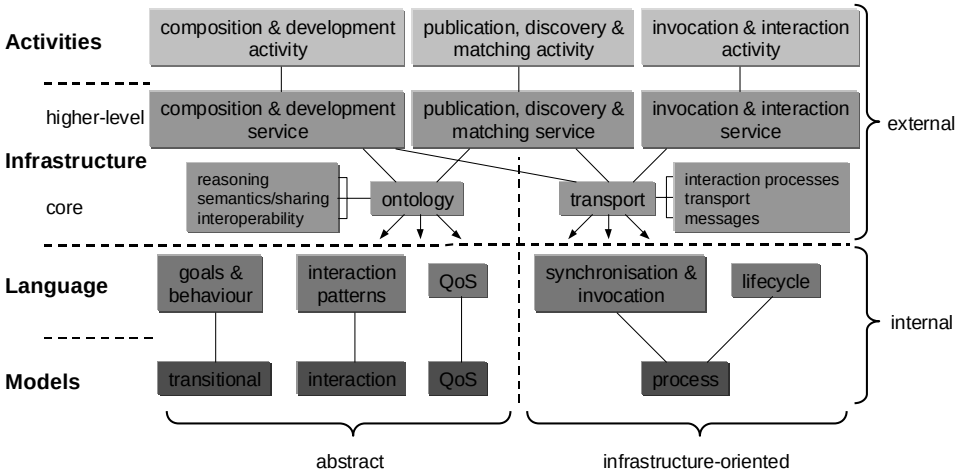

Fig. 1. A Layered Conceptual Framework for Semantic Web Services

services [3, 2, 16, 19, 4, 22, 18, 23, 21, 14, 20, 13, 15]. The WSF focuses on message format and message exhange mechanisms to provide and invoke services. In addition, various semantical properties of services are relevant for a service user, e.g. [24, 2, 11]:

- *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.

Semantical properties – including behaviour and dependencies – enable the reuse of services and their independent composition, resulting in a different development and deployment style for Web services. We will ignore quality-of-service attributes [21] – which can range from software attributes such as maintainability, security, and efficiency to aspects such as pricing.

2.3 Extending the Web Services Framework

In the WSF [1], a description language, WSDL (Web Services Description Language), is used to describe syntactical 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), 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.

However, there is no support for semantical descriptions or the composition of services, and (business) processes cannot be modelled. Some attempts have already been made to rectify this [25, 2, 14, 13, 15]. For instance, BPEL4WS [14] and WSFL [25] are Web service flow languages that allows Web services to be composed (choreography and orchestration); DAML-S (now also known as OWL-S) [2] is a Semantic Web-compliant ontology for Web service description, and the Web Service Modelling Framework WSMF [3] is an ontology-based modelling approach.

Taking the concepts of these approaches on board, we can identify essential aspects of Web services that should form core elements of a Web services descriptions:

- *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* 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 can distinguish profile information (what a service requires and provides), model information (how the service works), and binding information (how the service is used).

We propose to follow the route taken by OWL-S and WSMF towards an extended WSF and base our framework on ontology technology. Moreover, we add a new aspect [8, 9, 26]. Component technology aims at modular composition of software systems from self-contained, reusable components described by contract-based interfaces and explicit context dependencies. 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 architecture WSA [1] focuses on messages, i.e. sees the description of message formats and their exchange at the core of the architecture, rather than the effects that are caused by message exchange. We focus on service semantics in the context of service choreography and orchestration where dependencies have to be made explicit. The automation of activities such as discovery is a central aim in both cases[2]. Whereas the aim of the WSA is not

[2] We use the term 'activities' here instead of 'processes', which we will use later on in a more technical sense

to prescribe particular development approaches, we will introduce here specific models and techniques for construction, discovery, and choreography of services – a consequence of chosing a specific (ontology-based) framework for service semantics.

We will discuss the central framework aspects – model and language, infrastructure and activities – in the next two sections. The discussion will always refer to Fig. 1. We refer to the literature to indicate the origin of and rationale behind framework elements.

3 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 a conceptual services framework. Knowledge engineering becomes therefore a pivotal technology.

3.1 Ontology Technology

The Semantic Web initiative aims at making the Web more meaningful and open to manipulation by software applications [5]. 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 [7] provide the reasoning support. Integrated into a ontological Web framework based on OWL – the Web Ontology Language – sharing of ontologies becomes possible [6].

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 our conceptual framework.

3.2 Service Models

In particular computational aspects of service properties need to be based on appropriate models that underlie semantical description and reasoning.

In previous sections we have outlined the types of information needed to adequately represent service behaviour, including input/output behaviour, interaction protocols, and service composition and communication. Three types of computational models can address these aspects [14, 20]:

- *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 [27].

  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 is needed, 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 [12]. 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 [21]. Due to their variety, we neglect a detailled description here.

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. The service model acts as a conceptual model, outlining essential modelling requirements arising from the Web services context, but also as a semantical model in which descriptions can be interpreted. The modelling requirements expressed through these models have to be reflected in ontology languages. For some of these aspects, extensions of a classical ontology language might be required.

3.3 Abstract Service Description

We suggest an ontology language – at the core of a variety of semantic Web services approaches such as [3, 2] – to introduce a description notation for abstract Web service properties. We use a description logic here, which underlies a language such as OWL [28]. Description logics are based on the idea of defining a concept in terms of its properties in relation to other concepts.

Describing Services as Processes. Central to the modelling facet of the framework is to understand services as processes [14, 20]. Service processes and service-oriented composition have been identified as weaknesses in the current WSF. A process view allows us to include process interaction. Moreover, it helps us to formalise (and eventually automate) stakeholder activities. Consequently, ontologies describing service properties in their domain context need to focus on processes as the central entities.

Description logic [7] knows two basic elements. *Concepts* are classes of objects with the same properties. For instance, in an 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 $\top, \bot, \neg, \sqcup, \sqcap$ and $\rightarrow$ with their usual set-theoretic meaning. For instance, $PrivateCustomer \sqcup CommercialCustomer$ is the concept that describes the union of both customer 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 $\exists balance.Numerical$.

Different ways to model services have been suggested. In [2, 16, 22], services are represented as concepts, with properties associated through roles. In [11, 12], services are modelled as roles, interpreted by accessibility relations between states. Essential 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. Properties of these states (in different formats) can be expressed using roles.

- *Goals* are abstract specifications about service behaviour [3]. *Assumptions* are pre-state properties that summarise basic concepts 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. An 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 lodgment service, the sum transfered into an account plus the pre-state balance yields the post-state balance. Contracts are refinements of goals [3].

Often, extensions of a classical ontology language [7] are necessary to enable goal and in particular contractual specifications. In [11, 12], it is necessary to introduce names into role expressions in order to express parameters. An example is

$$\forall.lodgment \circ \underline{Sum}_N; postCond.equal(Bal, pre\text{-}Bal{+}Sum)$$

saying that a transitional role *lodgment* is applied to parameter name *Sum*, and that after execution the balance *Bal* is increased by *Sum* (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 the operators, e.g. ; (sequence), + (choice), ! (iteration) to support the *interaction model* [12]. For instance,

$$open; !(lodge + transfer)$$

expresses that after opening an account, money can be repeatedly lodged or transfered.

3.4 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 [14, 20]. So far only considered as transitions in a state-based systems, we need to consider both the requester and the provider of these transitions [29, 30]. For instance, an automation of accesses to UDDI repositories would require such a process communication view. A description notation will build up on the ontology language for abstract service description by adding process calculi elements.

- *Ports.* The operation names define ports that, if synchronised with another port from another 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 $\overline{op}\langle a\rangle$ for output (sending) following [29] instead of an abstract role expression such as $op \circ a$ that we have introduced in Section 3.3.

The expression

$$getBalance(bal); \overline{newBalance}\langle bal + ldg\rangle$$

based on the abstract expression

$$getBalance \circ \underline{bal}_N; newBalance \circ (\underline{bal}_N + \underline{ldg}_N)$$

expresses that the specified service receives input from *getBalance* that it uses internally and then returns the result $bal + ldg$ to the service client using port *newBalance*.

Service Lifecycle. A notation to express service process interaction can be used to formalise an *activity-based lifecycle view on services* – which leads us into the infrastructure and activity aspects of our conceptual framework. Addressing the complete software lifecycle is an essential aspect of software engineering

methodologies [31]. 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 the different type 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

$$\textsc{Pro}\ s_C(s_I); !(\ \textsc{Exe}\ s_I(a, s_R); \textsc{Rep}\ \overline{s_R}\langle f(a)\rangle\ )$$

with annotations for providing Pro, executing Exe, and replying Rep for the contract, invocation, and reply ports s_C, s_I, and s_R, respectively [11]. The interaction pattern expresses that, after a contract match, a service can be invoked and a reply can occur an arbitrary number of times. This would formalise the UDDI-supported matching of WSDL descriptions of Web services and their invocation using SOAP in the WSF [1].

4 Infrastructure and Activities

The core task of a SOA infrastructure is to facilitate service invocation, but it also needs to support other stakeholder activities. The basic requirements for our conceptual framework arises already from the architecture required for discovery and invocation in the WSF. Semantic description and composition services can be layered on top.

4.1 Infrastructure Services and Facilities

Infrastructure services and facilities can be divided into core and higher-level:

- *Core infrastructure services* are *transport* – the distribution technology – i.e. the layered infrastructure model, and *ontologies* – the knowledge and semantics technology – i.e. the layered ontology model.
- *Higher-level services* build up on core services. *Publication, discovery, and matching* – based on transport and ontologies service – support discovery based on semantics-enabled UDDI and WSDL. *Development and composition* – based on transport and ontologies service – support composition and choreography based on semantics and interaction. *Service invocation and interaction* – based on the tranport service – support interaction for service invocation based on e.g. SOAP.

These services are usually provided through suitable APIs. Tools such as repositories, brokers, composition engines, and protocols facilitate these services.

We will discuss the higher-level services supporting development and deployment activities now in more detail. Distribution is the a property of a Web services architecture. Both development and deployment activities take place in this distributed context.

The central activities of invocation and execution of Web services shall be based on a *layered infrastructure model* for *transport* [32]. 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 covered. The essential aspect are the *interaction processes* – defining the sequencing of send and receive operations.

4.2 Development and Deployment Activities

A Web services architecture needs to enable stakeholders (providers and requesters) to carry out development and deployment activities. Building up on core infrastructure services (transport and ontologies) activities such as discovery, composition, and invocation and interaction need to be facilitated. A simplified process based on discovery, matching and invocation/interaction activities can be modelled through a lifecycle protocol; see Section 3.4.

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* [33], 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 [34]. A provided service needs to be at least as good as the requested one. In an ontology language, a *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 [22]. 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 3.2).

- *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 [35, 33, 34].
- *Interaction protocols* – based on an interaction model. A notion of simulation can form the basis of matching [30].
- *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 [11, 12].

Subsumption allows us to capture widely used software development concepts such as specialisation and refinement. An enhancement could be achieved if another crucial relationship, the part-whole relationship, is addressed. For example, the meronymy concept falls into this category. It addresses parts standing for the whole – something that happens if a service operation is refered to, but the whole service is actually meant.

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 [7] or through the use of transition system and automata-based approaches for verification [36].

Composition and Development. Composition can be both a development-time and a run-time activity. Services can be composed to composite 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 [8, 9]. 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. Internet protocols provide the basic infrastructure. On top of these, service-specific protocols such as SOAP provide an RPC mechanism. The ontology-based interaction patterns describe the interaction behaviour of services.

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. 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.

5 Conclusions

Semantic Web services are now an increasingly important topic. Several directions – including domain modelling and composition – are currently investigated. However, two central problems remain. Firstly, a coherent, integrating conceptual framework is lacking. In particular, how to integrate the different aspects models, language, infrastructure and activities, is not adequately addressed. Secondly, services as processes is a notion that is central to enhance the Web services framework – and that needs to be made explicit in conceptual frameworks and service architectures supporting these frameworks. Service choreography and orchestration are two terms that capture the idea of business and workflow process definition based on service process composition.

A high degree of automation is a requirement for the future of the Web services framework – scalability and, therefore, the success of the framework

depends on it. Automation requires shared semantics in a distributed, heterogeneous environment for development and deployment. Ontology technology is a solution to these problems[3]. Ontologies are reflected in all facets of our conceptual framework – models, languages, infrastructure, and activities. Ontologies can capture the process-oriented view on services and provide the necessary features to support the corresponding activities.

The proposed conceptual framework is the result of an investigation into various approaches in the context, guided by our own work. 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 captures current developments such as the Web service framework WSF, OWL-S, BPEL4WS, and others and places these in the wider development context. The framework promotes the idea of a lifecycle-oriented approach to Web services development.

Our analysis of a number of semantic Web services extensions indicates progress towards a new methodology for 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 framework is different in many ways from the Web services framework WSF. Firstly, it exhibits characteristics of a component framework. Secondly, 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 a framework for these service components that connects the facets model, language, infrastructure, and activity based by coherent Web-based ontology and transport technologies.

One of the aspects that we neglected in our discussion are quality-of-service issues. They include various aspects including performance and security. Security in particular is of paramount importance. In [37], we have already explored the extension of the WSF by security requirements descriptions. However, the integration with infrastructure technologies requires more attention.

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, pages 279–291. Springer-Verlag, 2002.
3. D. Fensel and C. Bussler. The Web Services Modeling Framework. Technical report, Vrije Universiteit Amsterdam, 2002.
4. J. Peer. Bringing Together Semantic Web and Web Services. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.

[3] The technical aspects of the ontology framework we have presented here are only indicative of what is needed on the language and model side.

5. W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.
6. H. Kim. Predicting How Ontologies for the Semantic Web Will Evolve. *Communications of the ACM*, 45(2):48–54, Feb 2002.
7. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
8. C. Szyperski. *Component Software: Beyond Object-Oriented Programming – 2nd Ed.* Addison-Wesley, 2002.
9. I. Crnkovic and M. Larsson, editors. *Building Reliable Component-based Software Systems*. Artech House Publishers, 2002.
10. 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*, pages 306–318. Springer-Verlag, LNCS 2870, 2003.
11. C. Pahl. An Ontology for Software Component Matching. In *Proc. Fundamental Approaches to Software Engineering FASE'2003*. Springer-Verlag, LNCS Series, 2003.
12. 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.
13. P. Bouquet, L. Serafini, and S. Zanobini. Semantic Coordination: A New Approach and an Application. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 130–145. Springer-Verlag, LNCS 2870, 2003.
14. D.J. Mandell and S.A. McIllraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 227–226. Springer-Verlag, LNCS 2870, 2003.
15. 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.
16. S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. World-Wide Web Conference WWW'2002*. 2002.
17. World-Wide Web Conference WWW'2003. *Semantic Web Services Panel*. ACM, 2003.
18. L. Chen, N. Chadbolt, 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.
19. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Semantic Configuration Web Services in the CAWICOMS Project. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
20. J. Bitcheva, O. Perrin, and C. Godart. Cooperative Process Coordination. In *Proc. International Conference in Web Services ICWS'2003*. 2003.
21. S. Ran. A Framework for Discovering Web Services with Desired Quality of Services Attributes. In *Proc. International Conference in Web Services ICWS'2003*. 2003.
22. M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.

23. K. Sivashanmugan, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Proc. International Conference in Web Services ICWS'2003*. 2003.
24. F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, 28(11):1056–1075, 2002.
25. F. Leymann. Web Services Flow Language (WSFL 1.0), 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
26. C. Szyperski. Component Technology - What, Where, and How? In *Proc. 25th International Conference on Software Engineering ICSE'03*, pages 684–693. 2003.
27. Bertrand Meyer. Applying Design by Contract. *Computer*, pages 40–51, October 1992.
28. I. Horrocks, D. McGuiness, and C. Welty. Digital Libraries and Web-based Information Systems. In F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors, *The Description Logic Handbook*. Cambridge University Press, 2003.
29. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
30. D. Sangiorgi and D. Walker. *The π-calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.
31. I. Sommerville. *Software Engineering - 6th Edition*. Addison Wesley, 2001.
32. World Wide Web Consortium. *Web Services Architecture Definition Document*. http://www.w3.org/2002/ws, 2003.
33. A. Moorman Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Trans. on Software Eng. and Meth.*, 6(4):333–369, 1997.
34. R.J.R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.
35. Bertrand Meyer. *Eiffel: the Language*. Prentice Hall, 1992.
36. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier, 1990.
37. C. Li and C. Pahl. Security in the Web Services Framework. In *Proc. International Symposium on Information and Communications Technologies ISICT'03*. 2003.