Knowledge Infrastructures for Software Service Architectures

Claus Pahl,
School of Computing, Dublin City University
Dublin 9, Ireland
Tel: ++353 +1 700 5620
E-mail: Claus.Pahl@dcu.ie

Abstract: Software development has become a distributed, collaborative process based on the assembly of off-the-shelf and purpose-built components or services. The selection of software services from service repositories and their integration into software system architectures, but also the development of services for these repositories requires an accessible information infrastructure that allows the description and comparison of these services. General knowledge relating to software development is equally important in this context as knowledge concerning the application domain of the software. Both form two pillars on which the structural and behavioural properties of software services can be addressed. We investigate how this information space for software services can be organized. Focal point are ontologies that, in addition to the usual static view on knowledge, also intrinsically addresses the dynamics, i.e. the behaviour of software. We relate our discussion to the Web context, looking at the Web Services Framework and the Semantic Web as the knowledge representation framework.

1. Introduction

Software development has changed dramatically over the past decades. Software development has become a distributed, collaborative process based on the assembly of off-the-shelf and purpose-built software services – a process that has recently been influenced by the Web as a software development and deployment platform. This change impacts the information and knowledge infrastructures for these software services.

The selection of services from service repositories, the composition of services, and the development of services for these repositories requires an accessible information infrastructure that allows their description, discovery, and assembly. Organising the knowledge space that captures these descriptions is essential. Discovery and composition of software services to service-based software architectures based on these abstract descriptions have become central activities of a new approach called service-oriented architecture [1].

In a distributed environment where providers and users of software services meet in electronic marketplaces, knowledge about these services and their properties is essential. Providers need to describe the properties of their provided services. Potential user need to understand these descriptions and need to be able to formulate their requirements in terms of queries in a marketplace or repository system. A shared knowledge representation language is a prerequisite.

We will introduce an ontological framework for the description of software services that supports the discovery and composition of these services within the Web Services Framework [8]. Ontologies as shared representations of knowledge are ideally suited to support this endeavour [4]. We will introduce a layered ontological

modelling approach based on description logics (a logic underlying various ontology languages), i.e. a logic-based terminological framework.

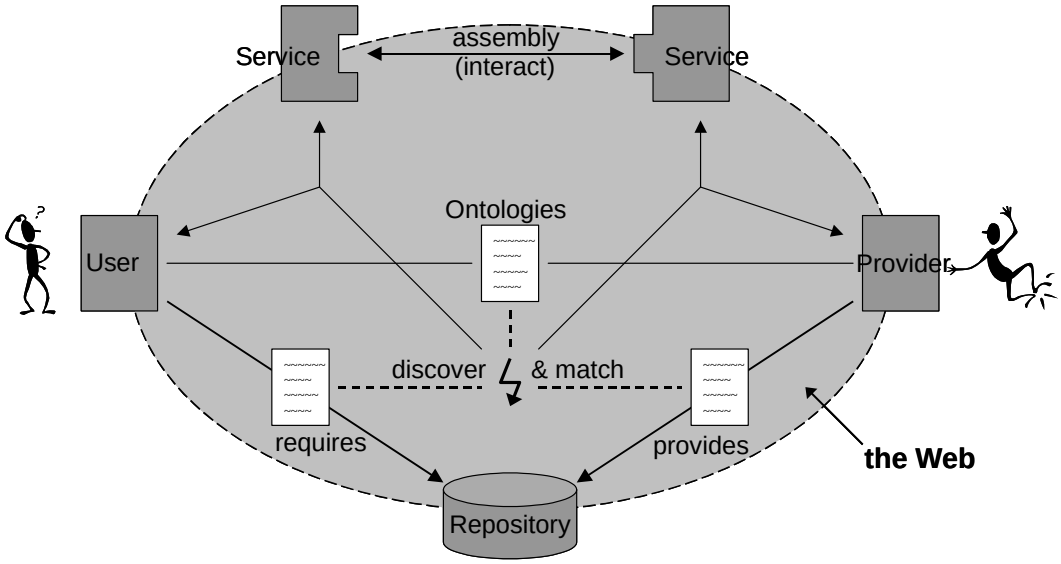

Figure 1. A Service Development Scenario for the Web.

2. Service-oriented Software Development

The World-Wide Web is currently undergoing a change from a document- to a services-oriented environment, i.e. the focus changes from information to computation. The aim of the Web Services Framework (WSF) is to provide an infrastructure of languages, protocols, and tools to enable the development of services-oriented software architectures on and for the Web [11]. Services are software applications that are provided 'as is' at particular locations. Service examples range from simple information providers, such as weather or stock market information providers, to data storage support and complex services supporting e-commerce or online banking systems. Service providers advertise their services; potential users can browse repository-based marketplaces to find suitable services, see Fig. 1. Once a suitable service has been found using a repository, a user can interact with the service directly to make use of its services. The prerequisite is a common language to express properties of these Web-based services and their application context. Ontology languages can provide this common language.

Modelling is an activity of central importance in this context. Various models are used in the development process:

- Computation-independent domain models capture the characteristics of the application domain.
- Platform-independent models describe the software system in abstract terms, e.g. as a service-based architecture.
- Platform-specific models relate more abstract models to the constraints imposed by a platform such as the WSF.

Ontologies are the basis of our layered modelling approach. Model-Driven Architecture (MDA) is an effort supported by the Object Management Group [7] related to our aims. MDA acknowledges the importance of modelling for architectural design of service-based software systems. We have used ontologies instead of the Unified Modelling Language UML – the OMG suggestion – for modelling. We have

based our layered ontology framework on the three model layers proposed by the OMG: computation-independent, platform-independent, and platform-specific.

The ontological modelling of services and their context is our central concern (Fig. 1). We will look at how these models are used in the software development process. Two activities are most important: discovery of provided services (lower half of Fig. 1) in structured repositories and composition of discovered services in service architectures through interaction (upper half of Fig. 1). For a software developer, this architecture means that most software development and deployment activities including search in external repositories and interaction with remote services will take place outside the boundaries of her/his own organisation – thus making shared knowledge essential.

3. An Ontology-based Knowledge Infrastructure for Services Development

3.1 A Knowledge Space for Software Services

The Web as a software platform is characterised by different actors, different locations, different organisations and different systems participating in the development and deployment of software. As a consequence, shared and structured knowledge about services plays a central role. A common understanding and agreement between the different actors in the development process is necessary. A shared knowledge space for software services in service-oriented architectures is needed. The question how to organise this knowledge space is our central question. In order to organise the knowledge space through an ontological framework, we address two facets of the knowledge space: firstly, the types of knowledge that we are concerned with and secondly, the representation of knowledge [10].

Three types of knowledge can be represented in three layers. The application domain is the basic layer. Abstract static and dynamic service properties form the middle layer. Platform-related knowledge for service development and deployment forms the last layer.

In general, knowledge representation [10] is concerned with the description of entities in order to define and classify these. Entities can be distinguished into objects (static entities) and processes (dynamic entities). Processes are often described in three aspects or tiers:

- Form – algorithms and implementation – the ’how’ of process description.
- Effect – abstract behaviour and results – the ’what’ of process description.
- Intention – goal and purpose – the ’why’ of process description.

We have related the aspects form, effect, and intention to software characteristics such as algorithms and abstract behaviour. Services are software entities that have process character, i.e. we will use this three-tiered approach for their description.

3.2 Ontologies

Ontologies are means of knowledge representation, defining so-called shared conceptualisations. Ontologies are frameworks for terminological definitions that can be used to organise concepts in a domain. Typical examples of ontologies are taxonomies, i.e. classification schemes used for example to classify animals or plants into hierarchies. Combined with a symbolic logic, we obtain a framework for specification, classification, and reasoning in an application domain. In a genealogy ontology, logic rules such as ‘the sister of a parent is an aunt’ complements the defined concepts such as parent, sister, etc. Terminological logics such as description logics [2] are an example of the latter.

The Semantic Web is an initiative for the Web that builds up on ontology technology and supporting knowledge engineering techniques [3]. XML is the syntactical format. RDF – the Resource Description Framework – is a triple-based formalism (subject, property, object) to describe entities. En example is (person, has_father, male_person). OWL – the Web Ontology Language – provides additional logic-based reasoning based on RDF.

We can use Semantic Web-based ontologies to formalise and axiomatise processes in a suitable logic, i.e. to make statements about processes and to reason about them. Description logic, which is used to define OWL, is based on concept and role descriptions [2]. Concepts represent classes of objects; roles represent relationships between concepts. Concept descriptions are based on logical combinators (negation, conjunction) and hybrid combinators (universal and existential quantification).

4. Description and Modelling of Services

Modelling of services and composed service processes is a stepwise process. Starting with a model of the underlying application domain, then individual services are modelled before, finally, their composition to business processes is addressed.

```
service AccountProcess
        operation   import Login (no:int,user:string) : bool
                    import Balance (no:int) : real
                    import Lodgement (no:int,sum:real) : void
                    import Transfer (no:int,dest:int,sum:real) : void
                    import Logout (no:int) : void
        process     Login; !(Balance+Lodgement+Transfer);Logout
service BankAccount
        operation   export Balance (no:int) : real
                    export Lodgement (no:int,sum:real) : void
                    export Transfer (no:int,dest:int,sum:real) : void
                    import CheckAcc (dest:int) : bool
        process     !(Balance+Logdement+(Transfer;CheckAcc))
service AccountRegistry
        operation   export CheckAcc (no:int) : bool
        process     !CheckAcc
service LoginServer
        operation   export Login (no:int,user:string) : bool
                    export Logout (no:int) : void
        process     !(Login+Logout)
```

Figure 2. An Online Banking Service.

Fig. 2 describes a central online banking process, defined in the AccountProcess service, that uses (or imports) other services to fulfil its tasks, i.e. AccountProcess is a client of BankAccount and LoginServer. The latter uses services provided by AccountRegistry.

4.1 Domain Models

Domain models form the starting point for many software developments. Central concepts of an application domain have to be identified and described in their properties (as relationships to other concepts). For the banking sector – see Fig.2,

which describe an online banking service – we would identify concepts such as account number or account user (which are static objects) and account login, lodgement, and transfer (which are dynamic activities or processes). In the context of software development, the capture of these objects and processes is particularly important. Processes for instance are described in terms of the objects they process. The resulting model is a semantic net consisting of (two types of) concepts and roles relating these concepts.

4.2 A Service Process Ontology

An intuitive approach to represent software behaviour in an ontological form would be to consider services as the central concepts [5]. We, however, propose a different approach that is particularly suitable for the abstract, platform-independent description of services and service processes. Our objective is to represent software systems. These systems are based on inherent notions of state and state transition. States of the systems will be the central concepts; transitions (services) will be represented as roles. Fig. 3 illustrates the central ideas. Service executions lead from old (pre)states to new (post)states, i.e. the service is represented as a role (a rectangle in the diagram). For instance, we could specify that a customer may check his/her account balance, or, that a transfer of money must result in a reduction of the source account balance. Usually, relationships in ontologies are used to express static properties, but they can also be seen as accessibility relationships between states of a system.

The transitional roles are complemented by more static, descriptional roles. For instance, preCond associates a precondition to a prestate; inSign associates the type signatures of possible service parameters. Some properties, such as the service name, will remain invariant.

Central to ontologies at this layer is the intrinsic specification of process behaviour in the ontology language itself. Behaviour specifications based on the descriptions of necessity and possibility are directly accessible to logic-based methods; behaviour-related inference of service properties is possible.

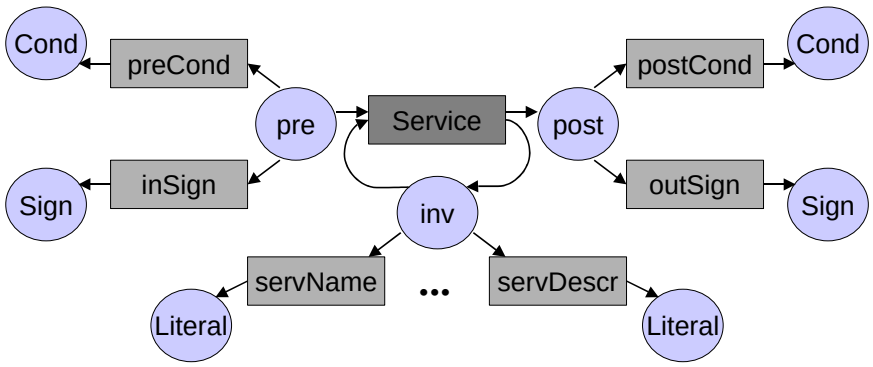

Figure 3. A Service Process Ontology.

4.3 Description of Services

Knowledge describing software services is represented in three layers.

- The intention is expressed through assumptions and goals of services in the context of the application domain. The domain model – see Section 4.1 – is the basis for these descriptions.

- The effect is a contract-based specification of system invariants, pre- and postconditions describing the obligations of users and providers. The process-oriented, platform-independent model – see Section 4.2 – captures this.
- The form defines the platform-specific aspects services. Platform-specific descriptions for the Web services platform consist of standardised, infrastructure-supported languages like the Web Service Description Language WSDL or extensions such as service ontologies like OWL-S [5] or WSMO [12].

A comprehensive framework would address the layers, but also transformations between them. We focus on modelling here, in particular on abstract, platform-independent effect descriptions, see Fig. 3. Effect descriptions are based on modal operators. These allow us to describe process behaviour and composition based on the choreography of service interactions. Composition in Web- and other service-oriented environments is interaction. Services are considered as independent concurrent processes that can interact (communicate) with each other. Central in the composition are the abstract effect of individual services and the interaction patterns of services:

- We introduce role expressions based on the role constructors for sequential composition *R;S*, iteration *!R*, and choice *R+S* into a basic ontology language to describe processes. Using this language, we can express ordering constraints for parameterised services. For instance, *Login; !(BalanceEnq + Transfer)* is a role expression describing an interaction process of an online banking user starting with a login, then repeatedly executing balance enquiry or money transfer.
- A logical effect specification focussing on safety is *positive(balance) -> Transfer . reduced(balance)* saying that if the account balance is positive, then money can be transferred, resulting in a reduced balance. Here, *Transfer* is the service; *positive(balance)* and *reduced(balance)* are pre- and postcondition, respectively. These conditions are concept expressions. Transfer causes a system to transfer from a prestate *pre* to a poststate *post*.

We use a connection between description logic and dynamic logic – a modal logic for the description of programs and processes based on operators to express necessity and possibility [6] – to address safety (necessity of behaviour) and liveness (possibility of behaviour) aspects of service behaviour. The central idea behind this connection is that roles can be interpreted as accessibility relations between states, which are central concepts of process-oriented software systems.

5. Discovery and Composition of Services

Service-based development is concerned with discovery and composition. In the Web context, both activities are supported by Semantic Web and Web Services techniques. They support semantical descriptions of services, marketplaces for the discovery of services based on intention descriptions as the search criteria, and composition support based on semantic effect descriptions. The actual deployment of services is based on the form aspect of process description.

5.1 Discovery

The aim of the discovery support is to find suitable provided services in a first step that match based on the application domain related goals and that, in a second step, match based on the more technical effect descriptions. This matching requires technical support, in particular for the formal effect descriptions.

- Service-based software systems are based on a central state concept; additional concepts for auxiliary aspects such as the pre- and poststate-related descriptions are available.
- Services are behaviourally characterised by transitional roles (for state changes) and descriptional roles (auxiliary state descriptions).

Matching can be based on techniques widely used in software development, such as refinement.

5.2 Matching and Composition

In order to support matching and architectural and process composition of services through ontology technology, we need to extend the (already process-oriented) ontology language we presented above [9]. We can make statements about service processes, but we cannot refer to the data elements processed by services. The role (or relationship) expression sublanguage needs to be extended by names (representing data elements) and parameters (which are names passed on to services for processing). We can make the *Transfer* service description more precise by using a data variable (*sum*) in pre- and postconditions and as a parameter: *balance >= sum -> Transfer(sum) . balance = balance@pre – sum* decreasing the pre-execution balance by *sum*.

Matching needs to be supported by a comparison construct. We already mentioned a refinement notion as a suitable solution. This definition, however, needs to be based on the support available in description logics. Subsumption is here the central inference technique. Subsumption is the subclass relationship on concept and role interpretations. We define two types of matching:

- For individual services, we define a refinement notion based on the design-by-contract principle, i.e. weaker preconditions (allowing a service to be invoked in more states) and stronger postconditions (improving the results of a service execution). For example *true -> Transfer(sum) . balance = balance@pre – sum* matches, i.e. refines *balance >= sum -> Transfer(sum) . balance = balance@pre – sum* since it allows the balance to become negative due to a weaker, i.e.less restrictive precondition *true*.
- For service processes, we define a simulation notion based on sequential process behaviour. A process matches another process if it can simulate the other's behaviour. For example the expression *Login; !(BalanceEnq+Transfer); Logout* matches, i.e. simulates *Login; !BalanceEnq; Logout*, since the *Transfer* service can be omitted. The provider needs to be able to simulate the process pattern requested by a potential user.

Both forms of matching are sufficient criteria for subsumption. Matching of effect descriptions is the prerequisite for the assembly of services in architectures and the composition of services to processes. Matching guarantees the proper interaction between composed service services.

6. Conclusions

Knowledge representation and management is increasingly important in all aspects of information technologies. Knowledge plays a particularly central role in the context of service-oriented software development. The emergence of the Web as a development and deployment platform for software emphasises this aspect. We have structured a knowledge space for software services in service-oriented architectures. Processes and their behavioural properties were the primary aspects.

We have developed a process-oriented, layered ontological model based on the facets form, effect, and intention. The discovery and the composition of process-oriented services based on ontological descriptions were the central activities. While some of the underlying techniques, for instance for matching, are already used in areas such as component-based software development, it is necessary to used widely accepted languages and techniques specific to the Web platform for Web services-based software development. Explicit, machine-processable knowledge is the key to future automation of software development activities. In particular, ontologies have the potential to become an accepted format that supports such an automation endeavour for the Web platform.

References

[1] ALONSO, G., CASATI, F., KUNO, H. and MACHIRAJU, V. (2004): Web Services - Concepts, Architectures and Applications. Springer-Verlag.

[2] BAADER, F., MCGUINESS, D., NARDI, D. and SCHNEIDER, P. (Eds.) (2003): The Description Logic Handbook. Cambridge University Press.

[3] BERNERS-LEE, T., HENDLER, J. and LASSILA, O. (2001): The Semantic Web. Scientific American, 284(5).

[4] DACONTA, M.C., OBRST, L.J. and SMITH, K.T. (2003): The Semantic Web – A Guide to the Future of XML, Web Services, and Knowledge Management. Wiley & Sons.

[5] OWL-S COALITION (2002): DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler (Eds.): Proc. First International Semantic Web Conference ISWC 2002. Springer-Verlag, Berlin, 279–291.

[6] KOZEN, D. and TIURYN, J. (1990): Logics of programs. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Vol. B, pages 789–840. Elsevier Science Publishers.

[7] OBJECT MANAGEMENT GROUP (2003): MDA Model-Driven Architecture Guide V1.0.1. OMG.

[8] PAHL, C. (2003): An Ontology for Software Service Matching. In Proc. Fundamental Approaches to Software Engineering FASE'2003. Springer-Verlag, Berlin, 208–216.

[9] PAHL, C. and CASEY, M. (2003): Ontology Support for Web Service Processes. In Proc. European Software Engineering Conference / Foundations of Software Engineering ESEC/FSE'03. ACM Press.

[10] SOWA, J.F. (2000): Knowledge Representation – Logical, Philosophical, and Computational Foundations. Brooks/Cole.

[11] W3C – WORLD WIDE WEB CONSORTIUM (2004): Web Services Framework. http://www.w3.org/2002/ws .

[12] WSMO Working Group (2005). Web Service Modelling Ontology (WSMO). http://www.wsmo.org/ .