Ontology-based Description and Reasoning for Component-based Development on the Web

Claus Pahl
Dublin City University, School of Computing
Dublin 9, Ireland

ABSTRACT

Substantial efforts are currently been made to transform the Web from a document-oriented platform for human consumption into a software-oriented application-to-application platform. The Web Services Framework provides the necessary languages, protocols, and support techniques. Even though Web services exhibit a basic component model, much can be gained by fully embracing software component technology for the Web. We propose to base this endeavour on ontology technology – an integral element of the Semantic Web. We will introduce an ontology that in particular provides a rich reasoning framework for behavioural aspects of Web services or, indeed, components on the Web.

1. THE WEB AND SOFTWARE DEVELOPMENT AND DEPLOYMENT

The Web is undergoing dramatic changes at the moment. From a human-oriented document publishing framework it has more and more developed into a platform where we can equally well find software applications. The application-to-application use of the Web is one of the recent endeavours to enhance the capabilities of the platform. The *Web Services* initiative [18] bundles these efforts to provide software applications in form of targeted services.

The current Web is a platform comprising *description languages* (such as HTML), *protocols* (such as HTTP), and *tools* (such as browsers and search engines) to support search, retrieval, transportation, and display of documents. The Web Services Framework provides a similar set of technologies – a description language WSDL (Web Service Description Language) for software services, a protocol SOAP (Simple Object Access Protocol) for service interactions, and tool support in form of UDDI (Universal Description, Discovery, and Integration Service) – a registry and marketplace where providers and users of Web services can meet.

Web services are important for middleware architectures. Various architectures, e.g. CORBA, have been established, but interoperability between these individual architectures, in particular in distributed environments, is still a major problem. Web services can, due to the ubiquity of the Web, provide interoperability.

Clearly, Web Services can encapsulate software components from various architectures and provide uniform Web-based interfaces and Web-based communication infrastructures for the component deployment. Web services themselves exhibit a simple *component model* [5, 16]. Even though service deployment has been the focus so far, the support of component-style deployment is, however, only one aspect of the Web Services Framework. We would like to emphasise here the importance of the development aspect – *component-based software development* using the *Web as the development platform* is often neglected or treated as a secondary aspect. Component development for the Web and using the Web requires the support by specific Web technologies that we will discuss here.

Besides services at the core of the Web Service Framework, we also look at current trends in this area. In particular Web service coordination creating service processes and interactions is an aspect that has received much attention [10, 1], and that is also of importance from a component-oriented perspective.

2. COMPONENT-BASED SOFTWARE DEVELOPMENT FOR THE WEB

Component-based software development [9, 16] is an ideal technology to support Web-based software development. As already mentioned, Web Services are based on a simple component model. A service encapsulates a coherent collection of operations. A central objective of component technology – central also for Web services – is the separation of computation and connection. *Computational aspects* are abstracted by interface descriptions. Usually, a set of semantically described, coherent operations forms an interface. *Connection* is more than plugging a provided operation into a requested operation – the coordination (or choreography) of services and their operations to more complex service processes is another important aspect of connectivity.

The (automated) interaction between component providers and clients becomes crucial in this context. Due to the nature of the Web platform, automation of these processes is highly important. The description of provided and required services needs to be supported. Reasoning to support matching of provided and required services is essential. Two aspects shall be distinguished:

- *Component connection and interaction* is based on plugging components together when a client requests services of a provider, possibly involving connectors or glue code. The semantical description of both provided and required component (or service) interfaces is essential in the Web context.

- The composition of services to *coordinated service processes* exhibiting a predefined *interaction pattern* is the other aspect [13]. Several extensions of the Web Services framework in this direction have already been made – such as the Web Ser-

SAVCBS'03 - ESEC/FSE'03 Workshop, Sept 1-2, 2003, Helsinki, Finland.

vices Flow Language WSFL [10] or the Web Services Coordination Language WSCL [1].

The architecture of this approach is illustrated in Figure 1. Critical for both forms is development support for a style of software development that is strongly based on retrieving and assembling suitable off-the-shelf components from a range of different component repositories. Reuse is the central theme for component development on the Web. The heterogeneity, distribution, and decentralisation requires a high degree of robustness and dependability of software components for the Web.

3. ONTOLOGIES AND COMPONENTS

3.1 Semantics and Knowledge

Describing the semantics of components means to express knowledge about the behaviour or other, non-functional aspects of a component or component process. This knowledge comprises:

- *Domain knowledge* from the application domain – basic entities and their properties are defined. This aspect – usually known as domain modelling in requirements engineering – is a widely accepted method.
- *Software-related knowledge* in relation to the type of semantics – behavioural semantics could express the behaviour of operations in a state-based system, techniques such as refinement could be defined. This is – due to the distributed and collaborative development on the Web – an emerging aspect.

3.2 The Semantic Web

In the context of *component-based Web service development*, the description of knowledge and also reasoning about knowledge are essential requirements. This requirement, in principle, can be satisfied through the techniques of another activity in the Web context – the Semantic Web initiative [18]. Description of and reasoning about knowledge is the objective of *ontology technology*, which is at the core of the Semantic Web. It provides XML-based knowledge description and ontology languages. Ontology languages facilitate the hierarchical description of concepts from a domain and their properties and support reasoning about these concepts and properties through a suitable logic.

The aim of the *Semantic Web* is to open the Web to processing by software agents. The exploitation of the full potential of the Web as an information network is currently hampered by the fact that information is provided for human consumption. Software processing, e.g. searches using search engines, is often inaccurate and error-prone. Adding semantical descriptions to Web resources and logic to reason about properties based on *ontologies shared between Web users* is the key contribution of the Semantic Web.

The application of ontologies is certainly not limited to application domains; they can also be used to capture software development principles and reasoning techniques.

3.3 Ontology Languages

The Semantic Web is based on the Resource Description Framework RDF – an XML-based language to express properties in terms of triples (Subject, Property, Object) [18]. Subjects (or concepts) are defined in terms of their properties in relation to other, already defined objects (concepts). We could, for instance, say that a component has an author, `(Component, hasAuthor, Author)`. In this way, based on some basic concepts, a hierarchy of complex, semantically defined concepts can be created. The ontology language DAML+OIL [18] (most likely the future Ontology Web Language OWL) is an extension of RDF by a rich set of operators and features to support ontology description and reasoning.

Reasoning is a central aspect that needs to be supported by a suitable logic. DAML+OIL is essentially a very expressive description logic. Description logics [2] are first-order logics that provide operators to specify concepts and their properties.

3.4 Ontology Support for Component-based Service Description and Composition

3.4.1 Description – Interface and Interaction

Ontology languages usually support the notions of concepts and properties (or roles)[1]. Ontology languages are application-domain independent. In the context of component-based Web services, the first essential question is what the description logic concepts and roles represent. An intuitive choice might be to use concepts to represent services or operations, and to express their properties using roles. We, however, suggest a different approach (Fig. 2). *Concepts* represent *descriptions of service properties*. *Roles* are used to represent the *services* themselves. Roles are usually interpreted as relations on classes of individuals (that represent concepts) – here they are interpreted as accessibility relations on states. This choice enables us to view a description logic specification as the specification of a state-based system with descriptions of state properties through concepts and specification of state transitions through roles. We actually distinguish two role types. *Descriptional roles* correspond to the classical use of roles as properties – examples in our ontology are preCond, postCond, opName, or opDescr, see Fig. 2. *Transitional roles* are roles representing operations, as we have just introduced.

Roles – supposed to represent services and operations here – are classically used in description logics to express simple concept properties. For instance, for a concept `Component`, the term $\forall$`hasAuthor.Author` is a *value restriction* that says that all components have authors. For a concept `State`, the *existentially quantified expression* $\exists$`preCond.valid(input)` says that for a given class of states, there is at least one for with a precondition `valid(input)` holds. Concepts are usually interpreted by classes of objects (called individuals). Both `hasAuthor` and `preCond` are roles – interpreted by relations on classes of individuals. $\forall$`update`.$\forall$`postCond`.`equal(retrieve(id),doc)` means that by executing operation `update` a state is reached that is described by postcondition `equal(retrieve(id),doc)`.

Even though some extensions of description logics exist in the literature [2], special attention has to be dedicated to roles if the needs arising from the application of ontology technology to components and services as suggested here have to be addressed. Elements of the language that need attention are: operations and parameters, service processes, and interactions of (provided and requested) services. We have developed a description logic that satisfies these criteria [14] – see Figure 2. At the core of this logic is a rich *role expression sublanguage*.

Operations – names and parameters: Usually, individuals can be named in description logics, but the notion of a variable or an abstract name is unknown. We have introduced names as roles, since they are here required as part of role expressions, interpreted as constant functions. Parameterisation is expressed through functional (sequential) composition of roles. We can express a parameterised role such as $\forall$`Login` $\circ$ `id`.`post` where `id` is a name that is a parameter to operation (role) `Login`.

[1]We focus here on description logic as the underlying ontology language – instead of the more verbose DAML+OIL.

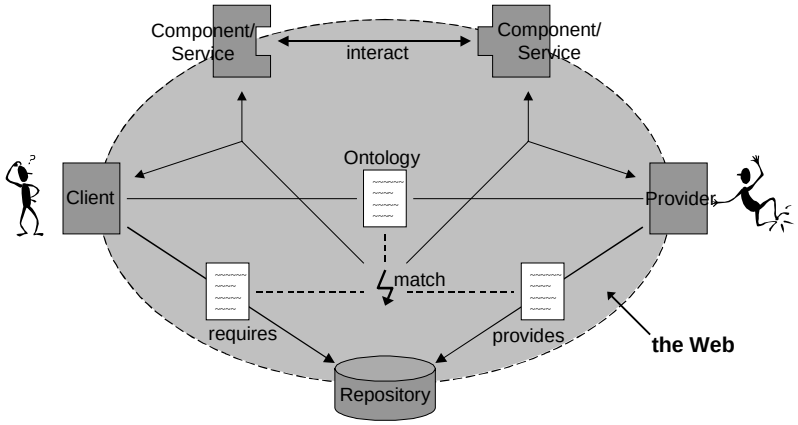

Figure 1: Ontology-based Component Development for the Web

Processes – role expressions: Besides the sequential (functional) composition of roles (representing operations and parameters), other forms of composition of services to service processes are required [14]. These composition operators include non-deterministic choice, iteration, and parallel composition; the following is an example $\forall$`Login; !(Catalog + Quote); Purchase.post`. The semantics of these operators can be defined in a transition system-based semantics for the logic.

Interaction – ports and the service life cycle: It is important to cover all aspects of the composition process including the matching of specifications, the connection of services, and the actual interaction between services [13, 14]. This is actually a central part of the life cycle of a service. A more network-oriented notion of ports, representing services in a component architecture is the central notion. Send and receive operations need to be distinguished – which means for the logic that role names can appear in these two forms.

The central aim of bringing ontology technology to component and Web service development is in enable meaningful, knowledge-level interaction through semantically decribed interfaces. Ontologies provide domain- and software development-related knowledge representation techniques for interfaces. Knowledge describing interfaces enables meaningful interaction.

3.4.2 Reasoning – Support for Service Matching

Reasoning in description logic is based on the central idea of *subsumption* – which is the subclass (or subset) relationships between classes that interpret either concepts (sets) or roles (relations). A composition activity that requires reasoning support in this context is *service or component matching*. The compatibility of a service requirements specification and the description of a provided service has to be verified.

Three research aspects are important in relation to matching support. Firstly, the subsumption idea has to be related to suitable *matching notions* for components and services. Secondly, the specific nature of *roles* and *role expressions* representing (composite) operations has to be addressed. Thirdly, the *tractability* of the result logical framework has to be considered. These research issues shall now be looked at in more detail.

Subsumption and matching: Matching between specifications of required and provided services can be expressed in form of classical notion used in computing, such as refinement or simulation [14]. For service interfaces we propose a *refinement* notion, which, if based on a design-by-contract matching notion on pre- and post-conditions, can be proven to imply subsumption [3]. For service process we propose a *simulation* notion on role expressions, which can also be proven to imply subsumption.

Role expressions and transitions: Roles are used to represent operations, i.e. roles have a transitional character. Essential is here suitable reasoning support for transitional roles. A *link* between *description logic* and *dynamic logic* (a logic of programs [7]) provides this support. Schild [15] investigates a correspondence between role-based concept descriptions in description logic and modal operators in dynamic logic. This correspondence allows us to adapt modal logic reasoning about state-based system behaviour into description logics. This correspondence is the essential benefit of chosing to represent services as roles, and not as concepts. Aspects of process calculi such as simulation notions can also be adapted for the description logic context [14].

Tractability: Tool support and automation are critical points for the success of the proposed component-based Web service development and deployment framework. Therefore, the tractability and a high degree of automation are desirable – even tough difficult to achieve. *Decidability and complexity* tend cause tractability problems in various logics. Here, we have proposed a *very expressive description logic*. Some properties of the proposed framework, however, seem to indicate that these problems can be adequately addressed. Transitional roles can be limited to interpretation by functions. Negation as an operator (known to cause difficulty) is not required for role expressions. Furthermore, application domain ontologies can be integrated through admissible concrete domains.

The specification of service and operation semantics often involves concepts from the application domain. Description logic theory [2] introduces a technique called *concrete domains* to handle the intoduction of specific classes of objects and their predicates into an existing semantics – an abstract example would be a number domain with its predicates. In order to retain the decidability in this framework, an *admissibility* criterion has to be satisfied. We have shown that this is possible for standard application domains such as numerical domains and domains of similar complexity [14].

4. RELATED WORK

DAML-S [6] is an ontology for Web services. DAML-S supports a range of descriptions for different service aspects from textual descriptions to behavioural semantics. The central difference between DAML-S and our framework it that DAML-S models services as

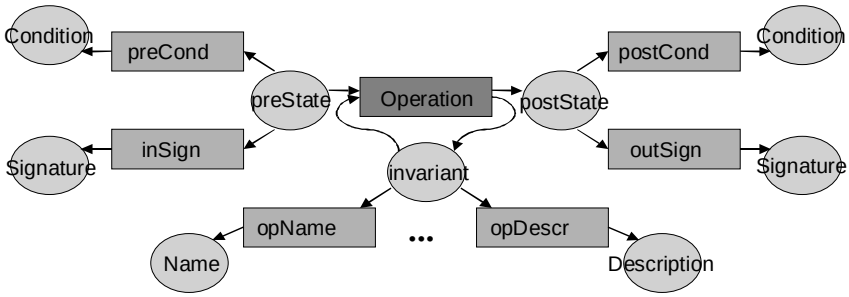

Figure 2: Service Process Ontology focussing on Operations

concepts, whereas we model services (more precisely operations of a service) as roles. The essential benefit of our approach is the correspondence between description logics and dynamic logic. This correspondence allows us to establish a richer reasoning framework in particular for the behavioural aspects of service descriptions.

The correspondence between *description logic and dynamic logic* was explored by Schild [15] around a decade ago, but has, despite its potential, not found its way into recent applications of ontology technology for software development. We have enhanced a description logic by results from two other software-engineering related areas – *modal logics* [7] and *process calculi*. Both have been used extensively to provide foundations for component-based software development, e.g. [11, 4]. For instance, advanced refinement and matching techniques [3] can be adapted for this context. Dynamic logic as an extension of Hoare logic can provide the framework. Design-by-contract is an important approach for the description of behavioural properties [12, 17, 8].

Description logic [2] relates to *knowledge engineering* – representing and reasoning about knowledge. The combination of knowledge and software engineering can result in fruitful outcomes.

5. CONCLUSIONS

Substantial effort is currently been made to make the Web a software development and deployment platform – the Web Services initiative. Component technology is ideally suited to support software development for the Web. However, this new application context also poses some challenges for component technology. Firstly, the Web is a ubiquitous platform, widely accepted and standardised. This requires component development techniques to adapt to this environment and to adhere to the standards of the Web. Secondly, the Web as a development platform is less well explored. As a consequence of distribution, decentralisation, and heterogeneity, the composition and assembly activities need to be well supported.

We have illustrated that technologies from another Web initiative – ontologies – can provide support for component-oriented Web service development. Ontology technology to represent and reason about knowledge can be adapted for components. We have explored the foundations for a composition framework for the Web. The framework is based on an ontology for components and services that incorporates reasoning support for behavioural aspects. An ontology – agreed and shared by developers and clients – can capture domain and software-technical knowledge.

Ultimately we aim to support flexible, collaborative, and adaptive component-based structures for the Web, ideally formed from federating, agent-like components. This would create an innovative, more autonomous software organisation. Of course, much work remains to be done until this vision is accomplished, but work also remains towards a fully implemented support environment. Aspects of automation will, if at all, be difficult to achieve.

6. REFERENCES

[1] A. Banerji et.al. *Web Services Conversation Language*. http://www.w3.org/TR/wscl10/, 2003.

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

[3] R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[4] A. Brogi, E. Pimentel, and A. Roldán. Compatibility of Linda-based Component Interfaces. In A. Brogi and E. Pimentel, editors, *Proc. ICALP Workshop on Formal Methods and Component Interaction*. Elsevier Electronic Notes in Theoretical Computer Science, 2002.

[5] F. Curbera, N. Mukhi, and S. Weerawarana. On the Emergence of a Web Services Component Model. In *Proc. 6th Int. Workshop on Component-Oriented Programming WCOP2001*, 2001.

[6] 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.

[7] D. Kozen and J. Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier Science Publishers, 1990.

[8] G. Leavens and A. Baker. Enhancing the Pre- and Postcondition Technique for More Expressive Specifications. In R. France and B. Rumpe, editors, *Proceedings 2nd Int. Conference UML'99 - The Unified Modeling Language*. Springer Verlag, LNCS 1723, 1999.

[9] G. Leavens and M. Sitamaran. *Foundations of Component-Based Systems*. Cambridge University Press, 2000.

[10] F. Leymann. Web Services Flow Language (WSFL 1.0), 2001. www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[11] M. Lumpe, F. Achermann, and O. Nierstrasz. A Formal Language for Composition. In G. Leavens and M. Sitamaran, editors, *Foundations of Component-Based Systems*, 2000.

[12] B. Meyer. Applying Design by Contract. *Computer*, pages 40–51, Oct. 1992.

[13] C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. In A. Brogi and E. Pimentel, editors, *Proc. ICALP Workshop on Formal Methods and Component Interaction*. Elsevier Electronic Notes in Theoretical Computer Science, 2002.

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

[15] K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. 12th Int. Joint Conference on Artificial Intelligence*. 1991.

[16] C. Szyperski. Component Technology - What, Where, and How? In *Proc. 25th International Conference on Software Engineering ICSE'03*, pages 684–693. 2003.

[17] J. Warmer and A. Kleppe. *The Object Constraint Language – Precise Modeling With UML*. Addison-Wesley, 1998.

[18] World Wide Web Consortium. *Web Initiaves*. www.w3.org, 2003.