Web Components and the Semantic Web

Máire Casey and Claus Pahl

School of Computer Applications
Dublin City University
Dublin 9, Ireland

Abstract

Component-based software engineering using the Web differs from classical software engineering. We investigate Web component engineering activites that are crucial for the development, composition, and deployment of components on the Web. The current Web Services and Semantic Web initiatives strongly influence our work. Focussing on Web component composition we develop description and reasoning techniques that support a component developer in the composition activities, focussing here on matching. We show how a component model can be integrated into a Semantic Web-style ontology for component development.

1 Introduction

Component-based Software Engineering (CBSE) [1], i.e. constructing software systems by composing components, is a form of software development that is ideally supported by the Web infrastructure. Distributed software development based on component selection from repositories and their integration coincides with the principle of the Web as a repository of information.

Efforts have already been made to make the Web suitable for the discovery and usage of software applications instead of documents. These efforts are bundled in the Web Services Framework [2]. However, software components on the Web are more than a collection of services. Component development in a distributed environment such as the Web requires additional support. The Semantic Web activity [3] can provide answers to problems arising from Web components. The Semantic Web aims to introduce meaning to the Web using ontologies and ontology languages.

Central activities in Web-based component development are the discovery of suitable components from component repositories and their integration into a software application. The process of matching is crucial for the composition and configuration of component architectures in a distributed environment. We will investigate how efforts of the Web Services and Semantic Web initiatives can be utilised for Web Component development. We will combine

c

Semantic Web techniques – ontologies and underlying logics [4] – with techniques widely used in foundations of software engineering – process calculi [5] and software architecture techniques. This will result in a matching and interaction framework for Web components within the constraints set by the Semantic Web.

Our objectives are to discuss the potential and importance of Semantic Web technology for CBSE. We outline central elements of a Web component model, their foundations, and how they relate to Web Services and Semantic Web techniques. These observations are supported by a recently conducted case study [6].

2   Web Components

The Web Services Framework (WSF) – which defines languages, services and protocols – is the attempt of the Web community to transform the Web from a document-centred to a services-centred environment and to open the Web to applications-to-application use [2]. We propose to develop a Web component framework that enhances the Web Services Framework and that allows successful software engineering technologies to be utilised on the Web.

The development and deployment of components using the Web requires, similar to the WSF, an infrastructure for the publication, discovery, composition, and interaction of components. Automation is a requirement that applies to all development activities. In particular a rigorous foundation for the semantic description, matching and interaction of Web components is essential. We assume the following characteristics of a component model [7,6]:

- *Explicit export and import interfaces.* In particular explicit and formal import interfaces make components more context independent. Only the properties of required services and components are specified.

- *Semantic description of services.* In addition to syntactical information such as service signatures, the abstract specification of service behaviour in interfaces is a necessity for reusable software components.

- *Interaction patterns.* An interaction pattern describes the protocol of service activations that a user of a component has to follow in order to use the component in a meaningful way.

3   Semantic Web

Making the Web more meaningful and open to manipulation by software applications is the objective of the Semantic Web initiative. Annotations expressing meaning help software agents to obtain semantic information about documents [3]. Knowledge representation and logical inference techniques form the backbone. For annotations to be meaningful for both creator and user of these annotations, a shared understanding of precisely defined annotations is re-

quired. Ontologies are the key to a semantic Web, expressing terminologies and semantical properties and creating the shared understanding. Ontologies consist of hierarchical definitions of important concepts in a domain and descriptions of the properties of each concept, supported by special logics for knowledge representation and reasoning. Web ontologies can be defined in DAML+OIL – an ontology language based on XML and RDF/RDF Schema [3].

Some effort has already been made to exploit Semantic Web and ontology technology for the software domain [8]. DAML-S [9] is a DAML+OIL ontology for describing properties and capabilities of Web services, which shows the potential of this technology for software engineering .

Formality in the Semantic Web framework facilitates machine understanding and automated reasoning. DAML+OIL is equivalent to a very expressive *description logic* [4]. This fruitful connection provides well-defined semantics and reasoning systems. Description logic is particularly interesting for the software engineering context due to a correspondence between description logic and dynamic logic (a modal logic of programs). We propose to define a semantic IDL and a reasoning technique for component matching in form of an ontology. The connection between description logic and modal logics allows us to introduce reasoning about component and service matching into the Semantic Web framework.

4   Semantic Web Components

Composition is the central component engineering activity. The prerequisite for a reliable and automatable matching and connection support for Web component composition is a framework that defines these activities precisely and that enables their implementation. Matching of a required and a provided component is based on the description of Web components focussing on semantic service description and interaction patterns.

4.1   Component Description

Component discovery and composition in a distributed and untrusted Web environment requires adequate descriptions of provided and required components and services. The description of Web components needs to cover a variety of functional and non-functional aspects. We focus here on two functional ones: *service behaviour*, captured through ports and port types, and *interaction patterns*, captured through component behaviour and life cycle specifications.

4.1.1   Services and Ports
The basic WSF building blocks are *ports*, which represent services. *Port types* define services based on input and output messages. We propose to extend

the WSF port type specification by contractual information capturing service semantics. A *service contract* consists of a signature and pre- and postconditions. We use pre- and postconditions as functional abstractions for ports, enabling the design-by-contract approach [10]. Modal logic is a logical framework that subsume the pre- and postcondition technique [11].

4.1.2 Component Life Cycle

We can identify several stages in the life cycle of a component such as service matching, connector establishment, or service invocation. A component can request a service using a contract port. The requested properties of the service can be provided as type information. These types – signatures and pre/postconditions – are constraints that determine whether a request for a service can be satisfied by a provider. If matching is successful, the component interacts with the service repeatedly by invoke the service at a port.

Composition activities can be captured in a standard life cycle form, a *component composition and interaction protocol*. Clients are parameterised by their required services. Client requests have to be satisfied before a connection is established and interaction can happen. Service providers need to deal with several clients at the same time. A provider does not need to engage in interactions with all its ports. A component is usually both client and provider, i.e. imports and exports services. Deadlock detection techniques can be used to discover mutual dependencies between components. Process calculi [5] are ideally suited to capture life cyles and interaction patterns.

4.1.3 Interaction Patterns

The process life cycles are standard forms that reflect general assumptions and constraints about the component activities – including matching, connector establishment and interaction. For a particular component, a more specific *interaction pattern* is usually required for a consistent usage of the component – for example often the provider requires a creation of an object to precede the object's proper use. Ignoring detailed connection and interaction constraints, we can simplify life cycle expressions to an *abstracted interaction pattern*. This abstraction is based on the assumption that the correct matching and interaction protocol is obeyed. *Import interaction patterns* describe how a client component expects to use imported services. *Export interaction patterns* describe how provided services have to be used.

We have introduced two forms of protocol specification. Firstly, the *instance level*, which is the detailed form including all aspects of the matching, connection and interaction process of component instances. Secondly, the *specification level* – the abstracted interaction pattern that describes an ordering of services that forms the basis of matching. We do not need a new notation to express the interaction pattern of a particular component. The abstraction of the life cycle form suffices. The advantage of the abstract form is that it can easily be integrated into a description/modal logic based matching ontology.

4

4.2   Composition: Matching and Connection

4.2.1   Interaction Pattern Matching

We define interaction pattern matching at the specification level, which is an abstraction of the full life cycle. The specification of interaction patterns describes the ordering of observable activities of the component process. We use a notion of simulation between processes to define interaction pattern matching between requestor and provider. We rely on the type system to express service matching, i.e. whether a provider port matches a request.

A client shows a certain import interaction pattern, i.e. a certain ordering of requests to execute provider services, called a *client interaction pattern*. A *requested interaction pattern* is a pattern that complements the client interaction pattern, i.e. the pattern that the client expects the provider to support. A provider interaction pattern *matches* a requested interaction pattern if the provider is able to simulate the request. This definition is about potential interaction. Therefore, we only consider complemented requestor patterns.

Successful matching can result in a connection between the components, which is formally acknowledged in form of a *contract*, consisting of the interaction pattern and the behavioural port types. Contracts have to be enforced. A provider, which matches the requested pattern, guarantees that the interaction pattern and the behavioural constraints of service activations are not violated. The simulation definition makes sure that for any particular service request there is a suitably matched provided service.

4.2.2   Service Matching

Two services described by their signature and pre- and postconditions and represented by contract ports *match* if the precondition is weakened, the postcondition strengthened, and a signature morphism can be constructed. This definition is derived from a dynamic logic inference rule – the consequence rule CONS. We have chosen dynamic logic as the framework because we can exploit the logic's expressive power to specify both safety and liveness properties. A second advantage will become clear when we discuss matching ontologies.

4.3   Ontologies

We have discussed foundations of a component matching technique. However, providing component technology for the Web also requires to adapt to Web standards. In our case where semantics are particularly important, ontology languages and theories of the Semantic Web approach need to be adopted.

4.3.1   Ontologies for Software Development

Ontologies are frameworks that define the concepts and properties of a certain domain, and provide the vocabulary and facilities to reason about these. Two ontologies are important for the Web component context:

5

- *Application domain ontologies* describe the domain of the software application under development.
- *Software development ontologies* describe the software development entities and processes.

The need to create a shared understanding for an application domain is long recognised. Client, user and developer of a software system need to agree on concepts for the domain and their properties. Domain modelling is a widely used requirements engineering technique.

With the emergence of distributed software development and CBSE also the need to create a shared understanding of software entities and development processes arises. We outline some basic elements of a software development ontology formalising Web component development, in particular providing the crucial matching support.

The starting point in defining an ontology is to decide what the basic ontology elements – concepts and roles – represent. Our key idea is that the ontology formalises a software system and its specification. *Concepts* shall represent component system properties. Importantly, systems are dynamic, i.e. the descriptions of properties are inherently based on an underlying notion of state and state change. *Roles* shall represent two different kinds of relations. *Transitional roles* represent services in form of accessibility relations, i.e. they represent processes resulting in state changes. *Descriptional roles* represent properties in a given state such as signatures, pre- and postconditions, and invariant descriptions like service name and description.

Constructors are part of ontology languages that allow more complex concepts (and roles) to be constructed. Classical constructors include conjunction $\sqcap$ and negation $\neg$. Hybrid constructors are based on a concept and a role – we present these in a description logic notation. The constructor $\forall R.C$ is interpretated as either an accessibility relation R to a new state C for transitional roles, or as a property R satisfying a constraint C for descriptional roles. Basic concepts are interpreted as states. Transitional roles are interpreted as accessbility relations.

4.3.2 Matching of Services

A service is functionally specified through pre- and postconditions, which can be expressed in a program logic such as dynamic logic. Matching of services has been defined in terms of constraints on pre- and postconditions, and has been represented through a subtype relation between contract ports.

Subsumption is the central reasoning concept in description logics. We can integrate reasoning about service/component matching into this approach. Subsumption – essentially a subclass relationship – shall be interpreted as a subset relationship on sets of states that satisfy pre- or poststate descriptions. Simple descriptional roles are treated in the usual style. If the pre- and postconditions are application domain-specific formulas, then an underlying

domain-specific theory, e.g. an application domain ontology, is needed.

4.3.3   Matching of Interaction Patterns

An ontology that captures interaction pattern matching requires an extension of classical description logics [4]. We need to look at process expressions on services. The following *role constructors* are introduced: sequential composition, transitive closure (iteration), intersection (parallel composition without interaction), and union (non-deterministic choice), see [4]. We can define a subsumption axiom expressing that P subsumes R based on a simulation for interaction patterns, i.e. P simulates R.

5   Related Work

Advanced services architectures for the Web have already been proposed. In [12], a component model underlying the Web services platform is identified. Modeling and composition of Web Services is currently investigated. The issue of composed Web services has also been addressed. However, these approaches do not address proper components. DAML-S [9] provides to some extend for Web services what we aim at for Web components. However, the form of reasoning and ontology support that we have provided here is not possible in DAML-S, since DAML-S services are modelled as concepts. Only considering services as roles makes modal reasoning about process behaviour possible.

Software architecture addresses problems that arise when systems are constructed from components. Components are identified as points of computation. Connectors define interactions between the compoments.

6   Conclusions

Component-based development needs composition techniques, i.e. support to discover, match, and integrate existing components into a system under development. We have presented composition techniques for semantic Web components that are interoperable with current Semantic Web technology. The Semantic Web will incorporate logic and reasoning, aiming at automation and unambiguous shared understanding. An important aspect is the adherence to Web standards, which provides interoperability with Web techniques and tools, and increases the acceptance.

We have been looking at matching of component descriptions on the specification level, resulting in a technique formalised as a matching ontology.

- An upper, more abstract layer supports matching techniques. Our central aim is the integration of description and matching techniques – addressing service properties and interaction patterns – into an ontology framework based on a connection between description logics and modal logic.
- A lower layer describes the life cycle and interaction processes of compo-

7

nent instances. It consists of observable behaviour of services, abstracted by modal logic formulas, which are incorporated via the type system into component process descriptions.

References

[1] C. Szyperski. *Component Software: Beyond Object-Oriented Programming – 2nd Ed.* Addison-Wesley, 2002. (to appear).

[2] World Wide Web Consortium. *Web Services Framework*. http://www.w3.org/2002/ws, 2002.

[3] W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.

[4] F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003. (to appear).

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

[6] M. Casey. *Towards a Web Component Framework: an Investigation into the Suitability of Web Service Technologies for Web-based Components*. M.Sc. Dissertation. Dublin City University, 2002.

[7] C. Pahl and D. Ward. Towards a Component Composition and Interaction Architecture for the Web. In *Proc. ETAPS Workshop on Software Composition SC2002*. Elsevier, ENTCS Series, 2002.

[8] 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.

[9] 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.

[10] Bertrand Meyer. Applying Design by Contract. *Computer*, pages 40–51, October 1992.

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

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