Towards a Component Composition and Interaction Architecture for the Web

Claus Pahl and David Ward

School of Computer Applications
Dublin City University
Dublin 9
Ireland

Abstract. The Web is currently undergoing a change from a document- to a services-centered environment. This shift can be seen as a first step towards a component-centered environment. We shall explore requirements for a Web component architecture based on the Web services framework, which has been promoted recently. A description language, protocols, and repository and directory services are the key elements. We will motivate an underlying conceptual model for these aspects capturing their foundations. We will identify two key features for a component architecture – a two-layered architecture and semantic descriptions of components – that makes it different from a services environment.

1 Introduction

The Web has evolved since its birth in the early 1990s. Originally designed as a publishing framework that allows users to make their documents available as hypertext documents and access other user's documents using a protocol that allows the transfer of hypertext document, it has evolved into a more dynamic and interactive environment. It is now used for purposes that were not intended at the beginning. The Web has become a bidirectional in terms of data transfer. In these days, a major evolution step is in progress, moving the Web from a document-centered environment to an application- or services-centered environment. Instead of accessing data, a user would be provided with the possibility to access services. This process should also enable application-to-application usage of the Web infrastructure. These attempts are focussed on individual services. The success of component technology [1] in recent years makes it worth while looking at component composition and interaction in a Web environment.

The focus of current research and development in Web technologies is on services – usually summarised by the term *Web services* [2]. We will explore here the use of the Web as an architecture for component composition and interaction. Instead of providing single services, several services are grouped into components encapsulating an internal state. In addition to providing services via an export interface, components also have an explicit import interface stating the services required by a component in order to work according to their specifications. Most approaches to component description suggest additional semantical

information to describe services. Contractual information in form of pre- and postconditions is a classical choice here. Requested and provided services have to be matched if components are composed to larger systems. Conformance rules describe the constraints governing the component matching. A second activity besides matching is the interaction between client and service provider. The activation of a component service is the same as for individual services, except that a component state might change.

The increased complexity of components for the Web – we will use the term *Web components* – with import and export interfaces and matching raises the question of an architecture for a Web components framework. Essentially, a *distributed computing model* for Web components is sought. The suggested architecture for Web services [2] consists of a services activation protocol, a services description language and a directory facility. If the Web services framework were to be extended to a Web components framework, we would need language support for semantical description in component interfaces, a protocol extended to two phases consisting of matching and interaction, and a set of services for lookup, matching, analyses, communication, etc. This paper aims at raising some issues in the development and standardisation of an architecture for Web components, and assessing the suitability of Web services and concepts from other frameworks such as CORBA for object technologies [3]. The ultimate aim is the development of a component composition architecture.

The extension from Web Services to Web Components has already been investigated in [4]. We carry their work further. We clarify the idea of a layered architecture, reflecting that composition of components consists of two phases: matching – sometimes called linking – and interaction. Technologies such as COM are concerned with interaction; module systems are concerned with linking. This idea is also advocated by the Cell-project [5] and in [6] – two approaches to components and the Internet. Another issue not considered in sufficient depth is the semantic description of Web components, which impacts the two-layered architecture and which has implications for possible services in such an environment. We will focus on synchronous interaction and put an emphasis on the description of components and matching between components. We will discuss some concepts and services supporting these issues, aiming at a clarification of critical issues.

We present principles of the Web services framework in Section 2. Then, we work out the shortcomings of this framework for Web component technology in Section 3. Section 4 describes key concepts for a formal model that can underlie a Web component architecture. This Web component architecture is then addressed in Section 5. Our focus is on the description language here. We end with related work and some conclusions.

2 Web Services – a Short Introduction

The purpose of the Web services framework is to move the Web from a document-centered environment to a service-centered environment. It aims to enable the

application-to-application use of the Web – the Web has so far been an environment used essentially by humans. Web technologies – languages and protocols – are used to provide a remote procedure call mechanism. The protocol shall be based on XML-messaging in order to achieve maximal interoperability.

Single services without semantical information can be described by the *Web Services Description Language WSDL*. A Web service description consists of five sections. An *abstract*, protocol-independent part consists of type, data and operation descriptions. The operation part – called 'portType' – describes the operations that implement the service functionality in terms of its typed input and output parameters. These parameters are described in a data part – called 'message'. Types for the messages can be defined in a separate 'types' section. The binding to a specific protocol is one of the two sections of the *concrete* part of the service description. It describes how a service is activated using the protocol under consideration. This section is called 'binding'. The final section is called 'service', and links the service to a particular location where the service can be found. The protocol used then determines the format to be used to activate a Web service.

The infrastructure for Web service activation and reply is given by the SOAP protocol – which might influence the standardisation of the XML Protocol [7]. SOAP – the *Simple Object Access Protocol* – is an XML-based protocol for service invocations and replies. It is designed to support remote activations of services specified using the WSDL. Discovery of services is supported by a directory framework UDDI – Universal Description, Discovery and Integration. UDDI acts as a marketplace for components.

3 Web Services – an Analysis

3.1 Services and Components

We have already pointed out some differences between services and components in the introduction. This discussion shall now be continued in detail. The following issues distinguish components from services in general – without looking at the Web environment in particular.

Import and export interfaces: Apart from services that are made available by components, component interfaces also describe services that are requested to fulfill the component's duties.

Semantic information: Services are described syntactically and semantically. The semantical description of services could be based on the design-by-contract approach [8]. An axiomatic description using the pre- and post-condition technique is possible.

Matching: Conformance between a requesting client and a service provider component – the provider matches the client requirements – needs to be considered. We can express notions of conformance through a type system. Type equivalence and subtypes can formalise conformance. A formulas-as-types approach for axiomatic semantic descriptions could be applied.

Dynamic configuration: A notion of connection needs to be introduced. When two components are composed, a private connection between client and server time needs to be created. Matching might or might not involve an agent or a composition broker. Connections between a service provider and a client can persist. The client can use the connection multiple times. Components might change their state as a consequence of service interactions. In evolving systems the spatial structure of component connections changes constantly due to new compositions and reconfigurations of single components (replacements) or systems of components.

Life cycle: Components need to be matched before any interaction can happen between the components, i.e. a protocol needs to be obeyed. Essentially, this is a two-phased protocol consisting of matching and interaction, but it needs to be extended if dynamic reconfigurations are considered.

3.2 Suitability of the Web Services Framework

The Web Services framework shall now be discussed in the light of the previous summary of component characteristics. We focus on descriptions of components here. We address the elements of Web services descriptions in WSDL.

Types: This element is based on a generic framework – the XML Schema language. Higher order connection types need to be introduced in order to capture the dynamic configuration of the spatial composition structure.

Messages: Messages can be of a connection type, reflecting that connection themselves need to be transported in order to create and change private connections between two components.

Port types: Port types need to be distinguished into in- and out-ports, and into matching and interaction ports. The latter types relate to the phase/layer, the former describe whether a service is part of an im- or export interface.

Binding: Several bindings for one service need to be introduced, such as matching binding and interaction binding.

Services: No change is needed compared to Web services.

An essential question, that has not been answered so far, is where the protocol or life cycle description has to be accommodated. Possibilities include the port types and the binding section. The Web services connections are once-off activations, whereas component connections can persist and might be used multiple times.

A major difference between the current Web services model and the distributed computing model for components that is sought, is that composition in the Web services framework is not ad-hoc. A component framework needs to cater for dynamic compositions. For components a process of agreement, e.g. based on contracts, is needed.

4 Web Component Architecture – Foundations

We shall now outline the elements of a conceptual model for Web components – essentially a requirements specification for such a model including formalisms such as type systems and transition systems.

4.1 Elements of a Core Model

Ports are abstract access points to component services. Port descriptions are part of interfaces. *Port types* can reflect various properties – e.g. the port polarity or orientation (input or output), the role (is the port involved in matching components or in the interaction of components), or the transport capacity. Port types can be used to express structural and behavioural constraints. A protocol endpoint (e.g. SOAP endpoint) is actually a family of ports with different roles.

The *type system* and in particular *subtypes* can play a major role. Subtypes can determine what a suitable match for a service request might be. The classical definition of a subtype [9] – an instance of a subtype can always be used in any context in which an instance of a supertype was expected – can formulate the essence of consistent matching.

The *composition architecture* is *layered*. We can distinguish a matching layer and an interaction layer. Connections for interactions are established after successful matching. These connections are needed for service activation and service reply. The connections can be private connections between components that persist for some period of time. This architecture is a reflection of the component life cycle. The *component life cycle* – matching before interaction – needs to be formalised by a *composition protocol*. This affects each component in isolation, but also the composition of components. Protocol constraints can be expressed by appropriate transition rules.

4.2 Advanced Concepts

Since we consider the Internet as the basic infrastructure for a Web component framework, some advanced aspects not covered in the core model come to mind immediately. These are *distribution, mobility* and *security*. The Internet is a distributed networking environment. Issues of distributed locations have to be addressed. Java is an example of an Internet programming platform that features mobile computation in form of applets. Security is certainly an issue in an open and distributed environment such as the Internet.

Another issue – not specific to the Internet, but very important – is *evolution*. Changing environments and requirements impact any kind of software system. We have already addressed dynamic reconfigurations in component systems.

4.3 Suitable Frameworks

A formally defined conceptual model for Web components is essential if analysis and reasoning services based on semantic descriptions shall be provided. Type systems and a notion of state-based transitions are crucial. Suitable frameworks for the formulation of this model are for instance process calculi with typing, mobility, security, etc – e.g. the π-calculus [10] or the Ambient calculus [11].

5 Web Component Architecture

An architecture for Web components should consist of:

- a *description language*: semantic component description
- a *matching and interaction protocol*: 2-phase (or 2-layered) composition
- a *set of services*: discovery, matching, configuration, replacement, interaction

Such an architecture would describe a component middleware platform. A formal model describing these languages, protocols and services has been suggested in Section 4. Description languages and protocols omit details about how components are discovered, how they are stored and made available. This can be supported by special services, such as a broker service. However, we shall address the essential element – the description language – only. A number of service will depend on the semantic formalism made available through the description language. Several supporting protocols might exist (cf. CORBA protocols GIOP and IIOP).

5.1 Web Component Description Language

Based on the conceptual model, a language for the description of Web components needs to be defined, called a *Web Components Description Language (WCDL)*. We will motivate this language by a schematic example – a full definition is beyond the scope of this paper – following the structure of the WSDL. Types – data types, port types, connection types – shall not be presented explicitly here. Important is the support of a subtype notion. Two messages shall be defined – a data item and a connection .

```
<message name="InData">
        <part name="body" element="dataType"/> </message>
<message name="InConnection">
        <part name="body" element="connectionType"/> </message>
```

Port types define the services based on these messages.

```
<portType name="service">
   <operation-contract name="servOp"
          precondition="..." postcondition="..." signature="..." >
        <input  message="..." type="connection"/>
   </operation>
   <operation-connection name="servOp">
        <input  message="..." type="data"/>
        <input  message="..." type="connection"/>
        <output message="..." type="data"/>
   </operation>
</portType>
```

The usage of these operations is expressed in form of a component life cycle – here a client requesting a service and then interacting with the service repeatedly:

```
<sequence>
   <request name="servOp" precondition="..." ... />
   <repeat>
      <sequence>
         <invoke  name="servOp"> ... </invoke>
         <receive name="servOp"> ... </receive>
      </sequence>
   </repeat>
</sequence>
```

The remaining sections of WSDL concern the concrete part, i.e. the protocol binding and association of the location. The binding part for WCDL needs to separate matching binding and interaction binding. The latter needs to address activation and reply. The service part addresses the location of the service. This part is not different from the WSDL.

5.2 Implementation

In order to study the feasibility of the concepts and ideas presented here, we have started implementing a prototype based on a central broker service [12]. This broker prototype is implemented on a standard Web-based 3-tiered architecture with a matching server and an interaction server. The matching server works based on a component repository, which contains only component interfaces – component executables themselves are located elsewhere. XML-based messaging is used to communicate matching- and interaction-related data. The broker includes an interface for matching to be used by a component system developer.

6 Related Work

Architectural frameworks exist for distributed object interaction – examples are CORBA or COM/DCOM [3]. We have in particular considered ideas from CORBA in our motivation of an architecture for distributed component interaction. CORBA-features such as method invocation, stubs/skeletons, services, and protocols have their correspondence in component technology.

The second kind of framework suitable for the Web components, that is discussed here, are Web services [2] – see previous sections for details. In [4], a component model underlying the Web services platform is identified. It is admitted that strenghtening the component aspects will greatly improve the platform. We have tried here to point out the shortcomings of that platform.

Some groups have already implemented component systems for the Internet. Among those are the Cell-project [5] and the ComponentXchange [6]. The former implements a two-layered system for component composition. The latter focusses on the matching activities – there called trading.

7 Conclusions

A framework for components on the Web requires more advanced features than the Web services framework or distibuted computing models for objects can deliver. Two aspects essentially make the difference. Firstly, matching (or linking) and interaction need to be separated, resulting in a two-layered architecture. Secondly, the presence of semantic descriptions increases the complexity, but also offers new opportunities that need to be supported by appropriate services.

We have suggested the development of a formal model that captures these concepts. A type system can formalise semantic descriptions and respective matching concepts. A protocol or transition system needs to formalise the separation of matching and interaction and the other life cycle constraints that apply. This formal model can form the basis of a Web component architecture.

The formulation of the underlying model or the definition of a Web component architecture is certainly beyond the scope of this paper. Our objective has only been to motivate their development and to point out essential concepts.

References

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

[2] V. Vasudevan. A Web Services Primer, 2001. http://www.xml.com/pub/a/2001/04/04/webservices.

[3] OMG. CORBA: Common Object Request Broker: Architecture and Specification, Revision 2.2. http://www.corba.org.

[4] 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.* http://research.microsoft.com/users/cszypers/events/, 2001.

[5] R. Rinat and S.F. Smith. The Cell Project: Component Technology for the Internet. In *Proceedings 6th Int. Workshop on Component-Oriented Programming WCOP2001.* http://research.microsoft.com/users/cszypers/events/, 2001.

[6] V. Sriram, A. Kumar, D. Gupta, and P. Jalote. ComponentXchange: A Software Component Marketplace on the Internet. In *Proceedings 10th Int. Conference on the World-Wide Web WWW10.* International World-Wide Web Conference Consortium IW3C2, 2001.

[7] W3C World Wide Web Consortium. Extensible Markup Language (XML), 2001. http://www.w3.org/XML.

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

[9] P. Wegner. Concepts and Paradigms of Object-Oriented Programming. *ACM OOPS Messenger*, pages 8–87, 1990.

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

[11] L. Cardelli and A.D. Gordon. Mobile Ambients. In *Proceedings FoSSaCS'98*, pages 140–155. Springer Verlag, 1998.

[12] D. Ward. Implementation of a Component Broker, 2001. Internal Project Report, School of Computer Applications, Dublin City University.