Conceptual Content Management for Enterprise Web Services

Sebastian Bossung, Hans-Werner Sehring, and Joachim W. Schmidt

{sebastian.bossung,hw.sehring,j.w.schmidt}@tu-harburg.de
Software Technology and Systems Institute (STS)
Hamburg University of Science and Technology (TUHH)

Abstract. Web services aim at providing an interoperable framework for cross system and multiple domain communication. Current basic standards are allowing for first cases of practical use and evaluation.
Since, however, the modeling of the underlying application domains is largely an open issue, web service support for cross domain applications is rather limited. This limitation is particularly severe in the area of enterprise services which could benefit substantially from well-defined semantics of multiple domains.
This paper focuses on the representation of user-specific domain models and on the support of their coherent interpretation on both client and server side. Our approach is founded on the paradigm of Conceptual Content Management (CCM) and provides support for coherent model interpretation by automatically generating CCM system implementations from high-level domain model specifications. Our approach to CCM has been successfully applied in several application projects.

1 Introduction

In their current state of development, web services implement means for interoperable communication between computational systems and their components. Such achievements are essential to most innovative systems. In the past there have been several architectural attempts to improve component interface definition and their safe use (DCE, CORBA). Although web services have brought several improvements by essentially lifting the matter from the component level to that of reusable and interoperating services, there remain numerous open issues. In particular, web services lack any substantial support for conceptual and semantic modeling since current web service standards deal only with technical aspects such as encoding of messages or addressing of certain operations.

Following the "find and bind" metaphor for web services, one of the major visions is the automatic discovery of services given the client's requirements. For this end, conceptual and semantic models of web services are needed, and if only for semi-automatic approaches to service discovery and safe service interoperability. Since in the case of enterprise web services the involved application models usually span several domains—human resources, finances, customer relations,

logistics, etc.—a shared model understanding and coherent use is particularly relevant for the web service providers as well as for their clients [3].

In this paper we propose to apply Conceptual Content Modeling (CCM) to the domain of web services expecting several benefits. For one, our approach allows an explicit declaration of the domain model concepts and entities (instead of declaring just technical types with little reusability, as in WSDL [17]). Secondly, it supports a coherent interpretation of such domain models by all participating parties through our generative CCM system implementation. Finally, personal extensions of domain models are possible, and support is given to integrate extended interpretations with each other. We have a prototypical implementation of web service support as part of our proven CCM technology.

The remainder of this paper is organized as follows: In section 2 we discuss the shortcomings of plain web service technology with respect to its interpretation of domain models. Section 3 gives a short summary of our CCM approach to domain modeling. Some of the requirements for enterprise services which we feel are not handled well by web service technology are discussed in section 4. In section 5 we present our support for web services by means of CCM. The paper concludes with a summary and some future research directions. Related work is discussed where appropriate.

2 Service Semantics

An important paradigm for dealing with computational artifacts is that of functional abstraction. Abstractions (also with respect to larger artifacts than functions such as classes or components) allow the creator of the artifact to pass along just the information that a user requires. Function signatures greatly enhance interoperability and are used within systems as well as for cross-system communication. Prominent examples of the latter are CORBA interface definitions and the Web Service Description Language (WSDL), but interface models achieve little in providing semantics for a computational entity.

In order to bring together objects in a computer system and their domain semantics, one needs to attach meaning to the former. One way to do this is through ontologies (e.g. [4] with regard to enterprises). These are essentially a vocabulary of terms with some specification of the meaning of each. Such terms have to be combined with the objects in a computer system they describe. Another possibility to describe the relationship of objects in a computer system to their application domain is the use of a conceptual modeling (see e.g. [2, 6]). We will introduce our conceptual modeling approach in the next section.

The web service standards that are in widespread use today (WSDL, UDDI, XML Schema, etc.) mainly deal with the syntax of service communications. Ontologies can be used to define the concepts a web service deals with. OWL-S[1] is an approach to modeling web services by means of an ontology. Modeling frameworks like WSMF (Web Service Modeling Framework, [5]) can then make

[1] http://www.daml.org/services/owl-s/1.1/

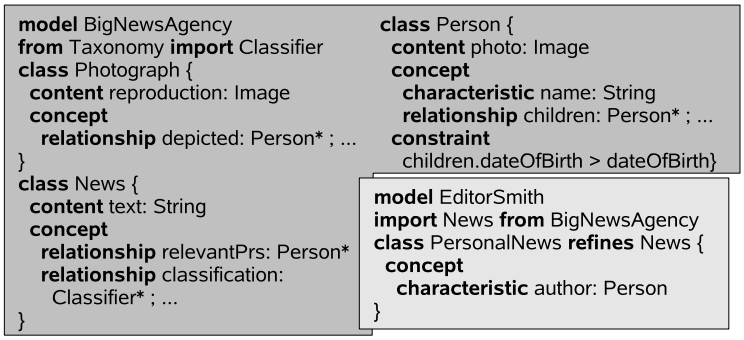

Fig. 1. Simple example of an asset model

use of this ontology to connect it to existing web service technology. While we basically agree with this approach, there are several problems with it, mainly centered around the way it causes semantics to be modeled. In particular we would like to emphasize that:

- The domain model is specified from the point of view of the web service. That is, input and output parameters of (possibly several) web services are given, causing a partial domain model to be implicitly defined by the union of all these parameters. Aspects that happen not to be needed for the collection of web services at hand are not modeled, resulting in an incomplete and difficult to understand "domain model".
- Reuse of such definitions is difficult [17], especially with regard to their incompleteness.
- Agreed-upon ontologies seem unlikely in the heterogeneous environment of web services, but subjective views of different communication parties will coexist (also see [9]). In the enterprise environment, this makes costly integration projects necessary [7].
- The proper implementation of the definitions (be they in ontologies or a commonplace web service standard) is still left entirely to the developers at both ends of the communication channel. Code generators for web services can do very little to enforce correct use of the domain model if all they know about this model are some fragments of XML Schema.
- In general, integration of ontologies into the rest of the system is not trivial [10], with regard to the amount of legacy systems found in usual enterprise environments this is even more evident.

In the next sections we detail how we deal with these problems in our Conceptual Content Management Systems (CCMSs).

3 Conceptual Content Management

Entities from the real world—concrete as well as abstract ones—are often described using content of various forms. However, these descriptions can only

be interpreted correctly with a conceptual understanding of the entity's nature. Therefore, entities must be described by a close coupling of content and concept, which we call an *asset*. In this section we will briefly introduce the key concepts of our asset language. More details can be found in [12–14].

3.1 Assets

Asset class definitions are organized in a model under the keyword `model` (fig. 1). They describe entities in a dualistic way by means of content-concept pairs. As an example, consider the definition of the class `Person`. In the `content` compartment a list of (multimedial) content objects is given. Possible handle types depend on the base language, which is currently Java. Content is treated as opaque by an asset-based system. It could for example be a photographic reproduction for a piece of art or an informal description (i.e., natural language text) in the case of more abstract entities.

The `concept` compartment provides a conceptual description of the entity. Characteristic attributes are those which are inherent to an entity (such as the person's `name` in the example), `relationships` are established between autonomous entities (e.g., the person's `children`). Note that attributes can be single or multi-valued, as indicated by the asterisk (`*`). Finally, constraints can be imposed on an asset class which are checked for all its instances. Constraints are defined over Java expressions in the case of characteristics, by relationship navigation, or by asset queries. Standard comparators (equal, less-than, etc.) are interpreted based on a type dependent meaning. Constraint expressions can formed by conjunction, disjunction, and negation.

Definitions from other models can be reused by importing them into a model (`from ... import ...`). They can then also be used as basis of asset class definitions, which is signified by the `refines` keyword.

3.2 Openness and Dynamics

Often—if not most of the time—different users will not be able to agree on an asset class definition to model entities from the real world. CCMSs acknowledge this fact by *openness* and *dynamics*. A system is open if its users are able to change the asset definitions in case they do not agree with what they find. A system is dynamic if it can react to such (re)definitions without requiring human (i.e. programmer) intervention.

CCMSs are both open and dynamic, which allows users to have their own subjective view of the world. Nevertheless, a CCMS also supports users in interacting with other users who might have a different model of the same entities. This is done with the help of mappings implemented in modules (section 3.3).

In the example of fig. 1 consider Mrs. Smith who is an editor at `BigNewsAgency`. She largely agrees with the definition of `News` by her employer, but would like to add a bit of information in her subjective view. To this end, she creates her own `PersonalNews` which also stores the `author`. By means of the

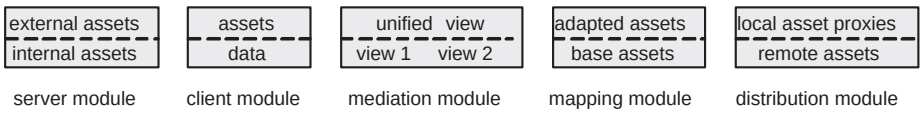

Fig. 2. CCMSs are built from different kinds of modules

CCMS she is able to do so without the help of a developer. The system reacts to her subjective view and incorporates it into its architecture (see next section).

3.3 Architecture

We have developed an architecture similar to the mediator architecture described in [16] which our Conceptual Content Management Systems use. This architecture makes use of five different kinds of modules arranged in layers and combined into components. We refer to such a combination as a system *configuration*. Fig. 2 gives an overview of the modules along with their interfaces. For more details please refer to [13, 14].

Such a system is generated by means of a compiler which is designed as a framework. The framework incorporates a number of generators that create the appropriate modules from asset definitions given by the user. The generators exchange information on their output by an extended kind of symbol table, and the framework synchronizes them according to their dependencies. Being available at runtime, the compiler enables a CCMS to react to user changes of asset definitions and thus plays a crucial role in implementing dynamics. Section 5 gives details on a compiler configuration for creating web-service-enabled systems (with an example in fig. 3). Sample system configurations for web services are also given in that section (specifically fig. 5).

4 Enterprise Conceptual Modeling

Enterprise applications work on large amounts of data which are crucial to their proper operation. These data need to be understood and modeled properly in order to enable the enterprise to do business. This section looks at three aspects of enterprise-wide models: cross-domain cooperation, personalization of domain models, and domain model evolution.

4.1 Cross-domain Cooperation

Developing enterprise applications consistently requires cross-domain cooperation. Each participating enterprise will typically have at least one model of its application domain [15]. This problem is reflected in the extensive literature on integration problems in enterprise applications (see e.g., [8]). To enable communication, it is beneficiary to have a conceptual domain model for each of the

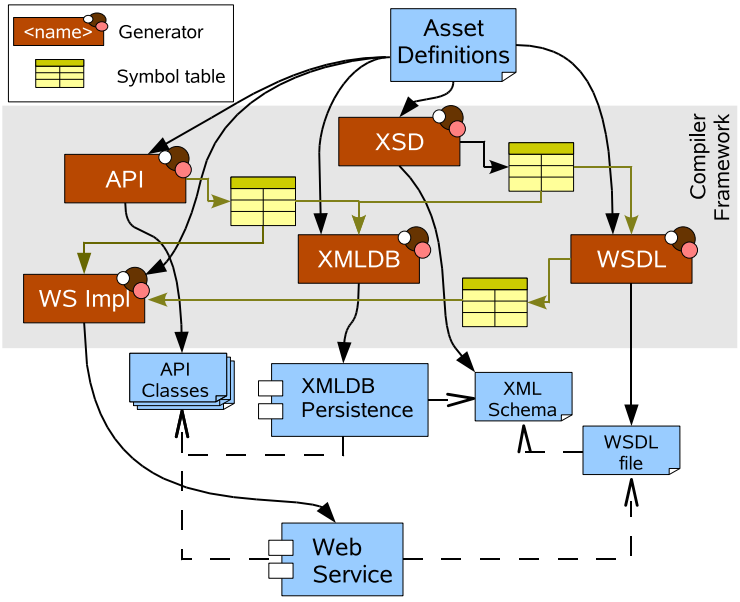

Fig. 3. Compiler Framework

partners. This greatly facilitates integration along the conceptual route and can even be combined with reasoning on ontologies, see [10].

4.2 Different Understanding of Domain Concepts

Even if two enterprises work in the same general area, they normally have slightly different understandings of their domain, yielding distinct but related models. In order to enable these enterprises to do business with each other, it is necessary to put their models into relation. There usually is a provider-requestor relationship between the business partners (e.g., a seller and a buyer). This aspect makes is feasible to describe one model as a personalization [11] of the other. A model of a company might have different personalization relationships with several models (importing from some, being the basis for others), which is not a problem when models can be composed of several sub-models.

4.3 Evolution with Respect to Domain Models

Most implementation technologies assume the domain model to be fixed, any modifications lead to a vast number of changes in almost all parts of the application. Nevertheless, one of the most common things to do for a domain model is to change. Reasons for this are manifold: Business circumstances might change, users might not agree to the opinion of others, or new use cases may arise. This calls for systems which support evolution of domain models in order to (a) allow changes in the first place (preferably in a way where most of the work is

```
model TabloidPress
import News from BigNewsAgency
class News {
  constraint text.length () <= 5000 and
    classification <= lookfor Classifier { name = "Triviata" }
}
```

Fig. 4. Personalized view of a client of BigNewsAgency

done automatically), and (b) provide clear semantics for such changes to enable backwards compatibility to the old system.

5 CCM for Enterprise Services

The requirements from the previous section are met by CCMSs whose modules are generated and configured accordingly. In our approach the basis for this is not a computational model of the web services' implementation, but a common domain model which describes the entities in the enterprise. The invocational details are left to standard web service technology.

Users wishing to set up a component participating in a CCMS use such a domain model or a personalized variant of it. The properties discussed in the previous section are available to asset definitions in form of the asset language. In this section we illustrate this by giving a small sample asset model. From such definitions CCMSs with the desired properties are generated. We will discuss some basic configurations of such CCMSs and argue their benefits.

5.1 A Sample Asset Model

Fig. 4 shows a model that is defined on top of the model `BigNewsAgency` introduced in fig. 1. In that model classes `News` and `Person` have been defined as part of the model of a news agency. From some other given model `Taxonomy` a class `Classifier` has been made available to classify instances of `News`.

A user working for Tabloid Press might use the model `BigNewsAgency` in conjunction with a new model `TabloidPress` shown in fig. 4. This model gives a new definition of the class `News`. For such news there now is a constraint which requires news to be at most 5000 characters long (using a method of the standard Java class `java.lang.String`) and to be classified under the `Classifier` "Triviata" (with the operator "<=" interpreted as "subset of", its default interpretation for multi-valued relationships).

When starting the asset model compiler it receives both the models `BigNewsAgency` and `TabloidPress`. It will generate modules for a configuration where the new system holds instances of `News`, `Person`, and `Classifier`. Having access to the original model the CCMS might be configured to interoperate with other systems which are based on the model `BigNewsAgency`, e.g., to take over existing `News` and to check whether they match the new definition. Likewise, `Classifier`s might be retrieved from a remote system.

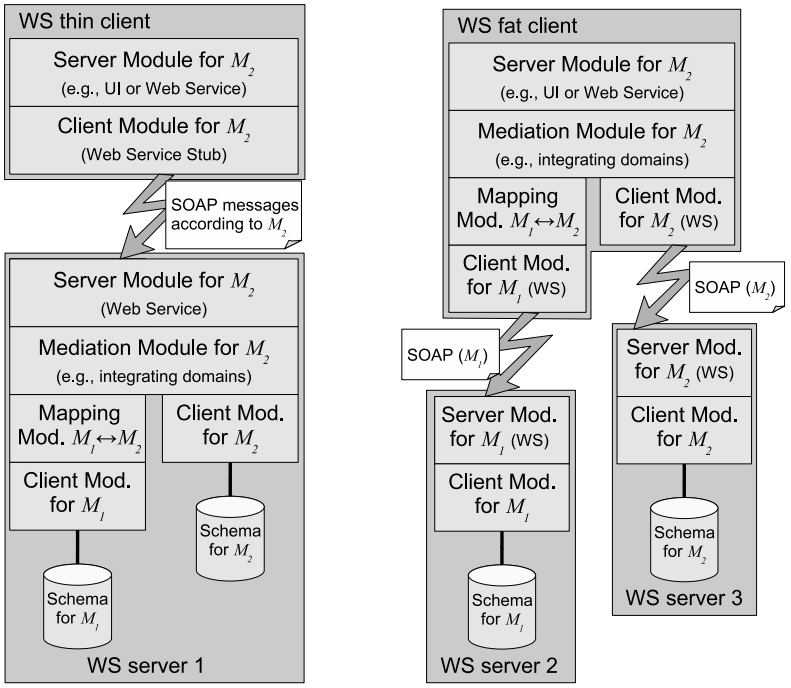

Fig. 5. CCMS configurations for enterprise services. (a) Three-tier configuration with application server and thin client, (b) Two-tier configuration with fat client

5.2 Sample CCMS Configurations

Fig. 5 shows two abstract sample configurations of CCMSs which meet the three requirements outlined in section 3—cross-domain cooperation, personalization of domain models, and their evolution. The asset models M_1 and M_2 shown in the examples represent two domain models. Depending on the demands to be met they can fill various roles as will be explained below.

Both systems outlined in fig. 5 show a client-server scenario incorporating two components—one offering a service, and the other using it. Both client components include a topmost server module by which the system is accessed. In the case of an interactive client this server module would be a graphical user interface, a web server, or the like. If the component is itself offering a web service the server module will implement a web service interface. The client components are based on a web service client module which sends request to components providing a web service.

The lower part of each of the systems shown in fig. 5 constitutes such service providers (WS server 1...3). Both include a server module to accept web service requests and a client module to access a third-party component. Instead of the

latter, web service client modules could be used if other services are used by the service at hand.

The system shown in fig. 5(a) shows a typical three-tier scenario in which a client component accesses a service on request. On the server side such a request is analyzed by a mediation module as part of the application logic and is delegated to a subsystem which handles either assets of a domain model M_1 or a model M_2.

Fig. 5(b) shows a configuration where the logic of using several servers and combining assets resides on the client ("fat client"). Depending on the assets involved, a web service request is sent to one of the server components WS server 2 or WS server 3 which host assets of models M_1 or M_2 respectively.

The asset models M_1 and M_2 shown in the examples represent two domain models. Depending on their relationship different scenarios are realized. Examples of models are:

1. M_1 and M_2 represent two domains and M_1 is integrated into M_2; assets from M_1 are possibly adapted before they can be integrated into M_2
2. M_2 is a personalized variant of M_1; assets in the public view represented by M_1 are lifted to M_2 by a mapping module
3. M_1 and M_2 are revisions of a model, where M_2 is the successor revision of M_1; by means of adaptation the assets from the outdated model are integrated into the newer version

Thus, the examples shown in fig. 5 exemplify CCMSs with the contributions outlined in section 4.

Other configurations of CCMSs than the ones indicated by fig. 5 cover different scenarios. E.g., assets from that part of the component WS server 1 that handles assets of M_2 could be adapted by an additional mapping module before being integrated into one model through the mediation module.

5.3 Benefits of Systems Generation

More complete systems than those in the examples from the previous section incorporate more layers in which mediation and mapping modules define the application logic. In enterprise applications there will be a number of components which use each other's services. The service to use is chosen by a mediation module as shown in the client component of fig. 5(b).

The key to the realization of the discussed system features—domain interoperation, personalization, and evolution—are the properties of openness and dynamics. These features allow servers and clients to share models of the domain at hand while still being able to change the definitions. This is especially important in enterprise systems as integration is expensive [7].

As pointed out in section 3 dynamics is achieved by generating all parts of a CCMS from asset definitions using the asset model compiler. As all components are generated on the basis of the same domain model, they interpret requests coherently. This ensures that modules which form components of a system are

created in a fashion that allows them to interoperate: Either they share the same domain model, or there are differences which some user stated explicitly. In the latter case mapping modules which allow the interoperation of components based on variants of a common domain model are derived from constraints describing model interrelations.

In the case of web services, generation covers interface definitions in WSDL as well as web service implementations in the form of server and client modules. The WSDL declarations contain XML schema information for the types involved which match the asset definitions. The web service operations are determined by the generators and implemented by corresponding configurations.

The asset model compiler framework has been introduced in fig. 3. This figure also shows several generators for the parts of a CCMS related to web services. A WSDL specification matching the asset class definitions is produced by the WSDL generator, while the server and client modules to provide and access services are created by the generator named WS Impl. The WSDL generator in turn uses XML Schema definitions created by the XSD generator which is also used, e.g., to define the schema of an XML database. As every module conforms to the uniform module API of CCMSs the generator for the web service server and client modules also uses the API definitions given by the API generator.

6 Summary and Outlook

To achieve interoperability through web services, semantic and conceptual models are highly important. We have shown that in this respect much can be gained by a coherent domain model which should be implemented in an open and dynamic way. Specifically, domain model entities should not just be modeled with respect to the service operations in which they are used. Supporting an application-wide approach through Conceptual Content Management has substantial benefits, which we have outlined in this paper.

In the future we plan to enhance web service support for CCMS in several ways. We will improve the coupling of web services (and possibly other technologies) with CCMS, for example in order to integrate legacy systems. Furthermore, the integration of standard web services (i.e., web services that are not implemented in a CCM enabled system) needs to be addressed, especially with regard to interoperability in an enterprise services environment. In a different area, we are also working on bringing together ontology-based semantic models, description logics and CCMS. We see two approaches to be pursued: firstly, description logics might be used to express the constraints in asset classes. The implications of changing semantics from closed-world to open-world will have to be explored. Secondly, with a mapping from asset class definitions to terms in ontologies [1] it would be possible to reason on the level of classes in contrast to performing model checking for given instances. This will require a mapping from asset classes to terms in ontologies. We aim to employ description logics to then perform reasoning in these systems.

References

1. Alex Borgida and Ronald J. Brachman. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter Conceptual Modeling with Description Logics, pages 349–372. Cambridge University Press, 2003.
2. Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Topics in Information Systems. Springer-Verlag, 1984.
3. Dov Dori. The Visual Semantic Web: Unifying Human and Machine Knowledge Representations with Object-Process Methodology. In *Proc. 1st Int. Workshop on Semantic Web and Databases*, 2003.
4. Fadi George Fadel, Mark S. Fox, and Michael Gruninger. A Generic Enterprise Resource Ontology. In *Proc. 3rd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1994.
5. Dieter Fensel and Christoph Bußler. The Web service modeling framework (WSMF). In *Database and Information Research for Semantic Web and Enterprises*, 2002.
6. Richard Hull and Roger King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3), 1987.
7. Jinyoul Lee, Keng Siau, and Soongoo Hong. Enterprise Integration with ERP and EAI. *CACM*, 46(2), 2003.
8. David S. Linthicum. *Next Generation Application Integration: From Simple Information to Web Services*. Addison-Wesley, 2004.
9. Peter Mika, Daniel Oberle, Aldo Gangemi, and Marta Sabou. Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In *Proc. of the 13th Int. WWW conf. 04*. ACM, 2004.
10. Boris Motik, Alexander Maedche, and Raphael Volz. A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications. In *Proc. of DOA/CoopIS/ODBASE 2002*, pages 1082–1099, London, 2002. Springer-Verlag.
11. Gustavo Rossi, Daniel Schwabe, and Robson Guimares. Designing personalized web applications. In *Proc. WWW 01*, pages 275–284, New York, NY, USA, 2001. ACM Press.
12. Joachim W. Schmidt and Hans-Werner Sehring. Conceptual Content Modeling and Management: The Rationale of an Asset Language. In *Proc. PSI 03*, volume 2890 of *LNCS*, pages 469–493. Springer, 2003.
13. Hans-Werner Sehring. *Konzeptorientiertes Content Management: Modell, Systemarchitektur und Prototypen*. PhD thesis, Hamburg University of Science and Technology (TUHH), 2004.
14. Hans-Werner Sehring and Joachim W. Schmidt. Beyond Databases: An Asset Language for Conceptual Content Management. In *Proceedings of the 8th ADBIS*, volume 3255 of *LNCS*, pages 99–112. Springer-Verlag, 2004.
15. René van Buuren, Henk Jonkers, Maria-Eugenia Iacob, and Patrick Strating. Composition of Relations in Enterprise Architecture Models. In *LNCS*, volume 3256, pages 39 – 53, 2004.
16. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25:38–49, 1992.
17. Jian Yang and Mike. P. Papazoglou. Web Component: A Substrate for Web Service Reuse and Composition. In *LNCS*, volume 2348, page 21, 2002.