**RC23819 (W0512-029) December 6, 2005**
**Computer Science**

# IBM Research Report

## Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05)

**Amsterdam, The Netherlands, December 2005**

**[1]Jen-Yao Chung, [2]George Feuerlicht, [3]Jim Webber (Eds.)**

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]University of Technology
Sydney, Australia

[3]ThoughtWorks
Australia

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Preface

While most observers agree about the benefits of service-oriented computing, there is less of an agreement about suitable architectural and interaction styles for Web Services applications. Of equal importance is the current lack of agreement about basic design principles for making decisions about what should constitute a service and its interface, and related considerations of service granularity. The transition towards service-oriented computing necessitates re-evaluation of design methodologies that are used in the construction of enterprise applications. Web Services design is an active research area and while there is some agreement about the basic design principles there are no comprehensive design methodologies for Web Services at present. There is a need for input from the research community and industry-based practitioners in order to develop design frameworks to support best practices for Web Services projects.

The focus of this workshop was on methods, frameworks and approaches for the design of service-oriented applications, and the workshop provided a forum for the discussion of design objectives, methods and guidelines for developing Web Services applications. Workshop discussion included other related topics such as design of industry-domain Web Services, design of Web Services for enterprise application integration, and methods for transforming existing component-based applications for SOA deployment.

The aim of the workshop was to bring together researchers and industry experts and provide a forum for exchange of ideas about design of Web Services and related issues. We have received paper submissions with topics including the architecture and design of service-oriented applications, engineering semantic services, designing reusable services, Web Service conflict management, and service discovery. All submitted papers were reviewed by three reviewers, and ten papers were selected for workshop presentations based on originality, relevance to the workshop topics, and overall quality. We thank the authors for their contribution to the workshop.

We thank all of the members of the Program Committee who were instrumental in ensuring the quality of the WDSOA'05 workshop, the organizers of the ICSOC conference and workshops, in particular Mike Papazoglou, Frank Leymann, and Winfried Lamersdorf for their help in planning and organizing the workshop. Special thanks to Christian Zirpins for his assistance with preparing the proceedings and collaboration in organizing the workshop, and Paco Curbera for his support in publishing the proceedings.

November 2005

Jen-Yao Chung
George Feuerlicht
Jim Webber

# Organization

## Workshop Chairs

Jen-Yao Chung, IBM T.J. Watson Research, USA
jychung@us.ibm.com

George Feuerlicht, University of Technology, Sydney, Australia
jiri@it.uts.edu.au

Jim Webber, ThoughtWorks, Australia
jwebber@thoughtworks.com

## Program Commitee

Andrew Blair, Biz Integration, Australia
Djamal Benslimane, LIRIS, France
Mark Cameron, CSIRO ICT Centre, Australia
Jen-Yao Chung, IBM T.J. Watson Research, USA
George Feuerlicht, University of Technology, Sydney, Australia
Ian Gorton, UNSW, NICTA, Australia
Paul Greenfield, CSIRO, Australia
Roy Grnmo, SINTEF ICT, Norway
Mark Little, Arjuna, USA
Mike Papazoglou, Tilburg University, The Netherlands
Savas Parastatidis, Microsoft, USA
Jiri Vorisek, VSE, Czech Republic
Jim Webber, ThoughtWorks, Australia
Andreas Wombacher, University of Twente, The Netherlands
Christian Zirpins, University of Hamburg, Germany

# Table of Contents

# SOA and the Future of Application Development

Bill Eidson, Jonathan Maron, Greg Pavlik, Rajesh Raheja

Oracle Corporation
224 Strawbridge Drive, Suite 300
Moorestown, NJ 08057
[jonathan.maron, greg.pavlik, rajesh.raheja, bill.eidson]@oracle.com

**Abstract.** Service Oriented Architectures (SOAs) signal a shift not only in the external facing aspects of application design, but also in the development of applications themselves. Most importantly, the network interface of business functions in SOA is structured at a higher level of abstraction than traditional distributed systems, focusing on the exchange of self-describing XML documents. These documents are often manifestations of canonical business events that are meaningful to higher-level business analysts. Utilization of these high-level constructs allows new services to be composed readily using technologies like Business Process Execution Language (BPEL), Enterprise Service Bus tools, and other XML-based technologies. With BPEL, for example, information exchange between business systems is often reducible to simple XML translations or transformations mediated by a process engine. Traditional systems programming is often relegated to the implementation of adaptor technology to interface with existing systems. We argue that this represents a serious evolution in the development of IT business solutions.

## 1 Characteristics of Traditional Distributed Programming

Traditional distributed application development approaches are primarily focused on low-level programming constructs such as sockets or programmatic support of object models as seen in CORBA, EJB, or DCOM. These approaches have a number of drawbacks. For example, socket programming is widely recognized as extremely low-level and error-prone [1]. Most business solutions delegate the management of network interfaces and message construction to middleware frameworks. The 90s were dominated by systems that used the object modeling paradigm for distributed systems.

Distributed object systems like CORBA or Java Remote Methods Invocation (RMI) are based on the idea of distribution transparency and provide a relatively simple approach to programming that models method invocation on an object without exposing whether the object is collocated in the same address space or distributed across the network. [2,3] Although distributed object models make the development of distributed systems simpler in comparison to developing custom message-over-sockets code, these systems still require application developers to expend a great deal of effort managing low-level system

constructs such as object lifecycle, fault tolerance, load balancing, security and transactions. In addition, developers are forced to struggle with the following problems:

1. Fault tolerance in distributed object systems is extremely challenging to manage. For example, failure modes are difficult to determine and object references are typically bound to a specific address space [4]. While many applications were built with distributed objects, the technology proved to be difficult to use as a basis for highly available systems without very specialized knowledge and experience. CORBA systems, for example, never successfully established a robust fault tolerance standard.
2. Object modeling tends to introduce a high degree of coupling in programs. With distributed objects, the coupling between clients and servers makes it difficult to think of participants as autonomous services, since the client must assume detailed knowledge of object interfaces and implementation details like lifecycle.
3. Because messages are passed as opaque encodings of programming language structures or native objects, it is difficult to effectively audit or process messages using intermediaries.
4. As a client-server model, distributed object systems do not offer an end-to-end message processing model, making distributed object systems less than ideal for structuring long-running business processes that execute over many systems.

Post-CORBA distributed object systems such as EJB address some of the issues in distributed object programming by providing containers that manage object lifecycle and system services. The EJB model, for example, manages object pooling, threading, security, distribution and transactions in the container to which the EJB is deployed. Many of these services are configured via XML deployment descriptors, though the application logic remains cognizant of the implications of the behavior managed by the container.

Despite the improved factoring in the EJB model, it suffers from the range of problems faced by developers using objects as a distribution model. As it is closely associated with the Java platform -- its invocation interface is based on Java types – EJB also faces interoperability difficulties. The latest generation of EJB systems is focused on providing a robust infrastructure for transactional business logic and persistence. EJB 3.0 is optimized for address-space-constrained Java objects (so-called “POJOs”). [5] This reorientation reflects the fact that EJBs are most often used as components within Web applications based on Servlet technologies. As a rule, this architecture does not require distributed object support.

## 2 Characteristics of Service Oriented Architectures

Service Oriented Architectures provide a very different model for applications. Rather than creating distributed applications by projecting object models across address spaces,

SOA applications create explicit boundaries for business functions, which are offered as services. A SOA is characterized by:

1. **Business functions that are explicitly modeled as networked services.** In contrast to distributed object systems, which emphasize distribution transparency, SOAs assume that service boundaries must be dealt with explicitly. Middleware can optimize for collocation scenarios, but a distributed architecture is assumed. This helps to address many of the problems and ambiguities associated with distributed object systems: in a SOA, both applications and infrastructure assume that system interoperability must be addressed explicitly and that network latency and failure may occur when crossing service boundaries.
2. **Application interfaces that describe the exchange of self-describing XML documents**. One immediate consequence of a SOA is an increased level of abstraction affecting the way in which services and their capabilities are described and consumed. The emphasis in a SOA is not on programming models; instead it focuses more directly on the exchange of business information to address specific business functions.

   The exchange of messages encoded in XML has several important advantages:

- Services can be built to process platform-independent data structures. This enables applications to more readily exchange data and build on common data formats.
- XML documents, particularly those conforming to the SOAP processing model, can be processed by intermediaries without prior knowledge of the XML schema used to define the business messages. Data-driven functions like content-based routing, auditing, and security can be implemented within the network. This is already widely supported by Web services management products.
- Both partial processing and transformation of data can be supported using readily available XML tools. Backward-compatible evolution of XML schemas for business documents also provides additional resiliency as services evolve independently.

3. **Well-understood meaning for messages based on canonical business documents or events**. With well-understood message definitions, services can more readily be integrated since they are designed to share a common information model. Legacy systems or pre-existing services can be integrated by translating from a canonical business message to a service specific message format. To further reduce coupling between systems, event driven capabilities are being introduced as a complement to SOA. Event-based service integration relies on a publish and subscribe mechanism to send the same message to multiple services for processing in parallel. A business event is defined by a canonical business document.
4. **Coarse-grained business functions that provide a specific service in isolation from orthogonal business functions**. Providing services with a well-defined and limited scope maximizes the reusability of services in new applications and processes. Because services tend to offer singular functionality, they are often implemented as stateless entities. One way to visualize a service is as a building block for more complex business applications. The SOA properties that we describe in this section are intended to complement this aspect of a service.

5. **Robustness of applications is increased by leveraging the ubiquitous protocols of the Web**. For example, service endpoints can be identified by URLs and SOAP is most commonly sent over the HTTP protocol. These characteristics provide several important advantages for SOAs. First, URLs are more easily redirected as infrastructure evolves because DNS is able to act as a universal directory. Second, Web protocols are better suited to support horizontal scalability for business functions and effective redirection and caching capabilities. Because SOAs are designed to provide flexibility in the carrier protocol for XML or SOAP messages, they are able to provide optimized channels for message distribution. For example, an XML business event can be sent over JMS for intranet subscribers and directly over HTTP to consuming applications in another network domain.

## 3 Impacts on Application Development

While SOAs offer a new paradigm for building applications, there are two key areas in which traditional systems development is still required: primitive business functions and adapter technology. New methods of service development will be used to build composite and interconnected services. We explore these three service types and how they relate in a SOA in this section

### 3.1 Primitive business functions

Middleware is normally used to expose business logic in a distributed environment. Because middleware manages many of the low level details of systems programming, business developers are free to focus on the function of their business service and the data it uses rather than issues related to network connectivity, transactions, etc. Development technologies such as EJB provide good technology support at the container level, but only provide developers an empty canvas. Lack of structure results in proliferation of non-uniform service definitions and additional effort in testing and certification throughout the development lifecycle.

We believe that developers of low-level distributed service components will benefit from frameworks that provide additional structure for service design. The important structuring aspects for service development are:

1. Consistent programmatic interfaces so that multiple business functions can operate in a homogenous fashion. Ideally, business services in this scenario will be implemented based on a-priori knowledge of the business documents that it will exchange. Standards such as Service Data Objects (SDO) are being designed to support this approach.

2. Built-in support for common design patterns such as factories are needed so that extensibility support is available to easily modify services in the face of changing requirements. We expect that a development framework will provide automated dependency resolution or so-called Dependency Injection.
3. We also strongly subscribe to the notion of tagging extensive metadata to the business functions. This should be supported via programming language annotations and XML configuration. This combination both simplifies development and leads to more flexibility in the solution, particularly if protocol bindings and policies are maintained independently from business logic. Metadata describing services and their capabilities can also be published to a central repository and queried with taxonomies that can be understood by business analysts.

The Service Component Architecture (SCA) is a framework jointly defined by BEA, IBM and Oracle that provides these capabilities for developers working in a SOA environment. SCA is described in detail in [6]. We believe frameworks of this nature will be increasingly important for implementing basic building block services.

### 3.2 Adapter technology

Traditional systems programming also plays a significant role in providing the code to link the service layer to legacy systems. The specific techniques for integrating a legacy business function into a SOA will vary based on environment. Some systems will require custom adapters; development of custom adapters often requires detailed knowledge of both the legacy system and the middleware functions of the system hosting the adapter. The Java Connector Architecture[7] is an example of an adapter framework that can be used for building adapters to connect back end systems into a SOA.

In many cases, adapters for legacy systems and enterprise applications are provided as packaged solutions by middleware vendors. In this case, bridging into a SOA may be as simple as applying transformations on the XML generated by the backend system into a business event defined for the SOA.

This combination of adapters and XML processing is useful for either creating new services on modern application server platforms or accessing business functions in legacy systems. They serve to expose existing business functions as services. Once these basic services have been defined, SOAs can be leveraged to compose new business functions by rapidly combining existing services into composite services.

### 3.3 Composite Services

The most significant change in development occurs in the way in which composite services are created and connected. Once the building blocks for a SOA are established within an IT organization, composite services will be the normal model for developing

new business functions or processes. As a rule, composite services will not be created in traditional programming languages. The foundational technology for implementing composite service development is the Business Process Execution Language [8], an orchestration language for combining XML messages from a SOA into new business processes.

The BPEL standard provides the following features:

1. Composite services are modeled as business flows. These new business processes are themselves simply exposed as Web services available within the SOA
2. Business interactions are driven by the exchange of XML business documents. BPEL process definitions themselves rely on XML as the basis of a process's data model.
3. Data exchange and process flow is defined primarily by reference to content in the XML business documents consumed by the process. Simple data referencing mechanisms like XPath are exploited for extracting data fragments used in decision trees.
4. Asynchronous interactions are enabled by the underlying process engine, which manages message correlation and process state. The inherently complex programming required to manage long-running business processes and data exchange is eliminated by the use of the BPEL engine.

Although an extensive examination of the BPEL syntax is beyond the scope of this paper, some of the most relevant constructs to this discussion include:

1. The `<scope>` element allows for the definition of a reversible unit of work. This allows for the definition of data variables as well as the specification of error handling and compensation handlers for the work performed within the given scope.
2. The `<flow>` parallel control construct allows multiple services to be invoked concurrently. There is no development effort required to handle the threading and asynchrony issues a concurrent invocation required in the past.
3. The `<receive>` element provides a facility for blocking while waiting for the matching message to arrive; the developer is not required to model the network message exchanges. The BPEL process manager further allows for the correlation of the responses once they are received; the processing within the scope of the `<flow>` element is not completed till all responses are received.
4. The `<sequence>` element allows for the ordered processing of the nested activities (service invocations, message receipts, etc.). Such standard control flow and branching activities are available to explicitly script the logic driven by the exchange of business documents as the process progresses.
5. The `<assign>` element uses XPath to copy fragments of XML between documents.

Many BPEL engines allow extension points to apply transformations via XSLT or other XML manipulation techniques.

BPEL provides a straightforward syntax for implementing concurrency, state management and control flow that is difficult to model in system programming languages like Java and C.

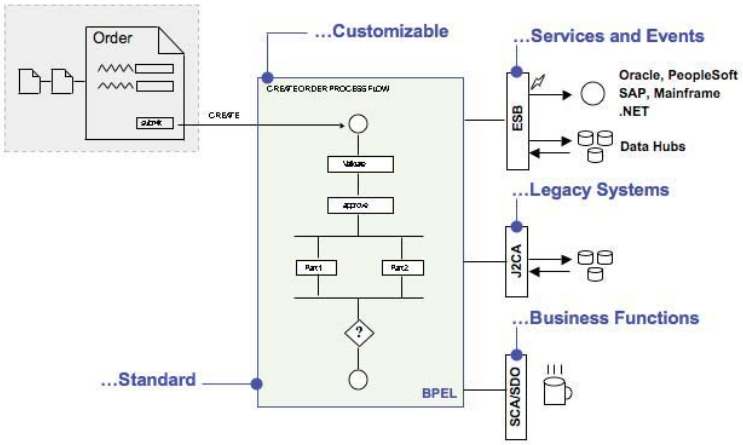


**Fig. 1.** Composite Services

### 3.4 Connecting Services

Concrete relationships are established by binding entities during the deployment of services into a SOA. In many cases, the coupling between services is reduced by using intermediaries as message routing agents. Canonical business documents at the core of a BPEL process are critical to allowing easy data exchange: if the same data structures are shared by many services, multiple services can more readily produce and consume shared information formats. These standard formats also enable event-driven relationships in which business events of interest, described by canonical message definitions, will be propagated to many services simultaneously for processing. The services within the application expose a number of business events that are raised at significant moments in the applications. For example, the application's customer module raises a new-customer business event whenever a new customer record is created. Other application modules can then add custom routing rules to the business event, without needing to modify the customer module itself. A routing rule can be used to invoke any service, including initiating a BPEL business processes. Similar to how modules within an application integrate through business events, routing rules can also be added for integration with external applications. These rules can transform and route the message to an external Web service, or to a service implemented by a legacy system adapter.

In the Oracle ESB, for example, an event source initiates a routing service to deliver the document to its appropriate destination, regardless of the network protocol. The difference in document formats from the canonical business event to the destination can be a simple XML transformation in certain A2A cases, or can be a complex long-lived business process in certain B2B standards, such as RosettaNet. Changing business requirements - such as the outsourcing of a given service - can be achieved by altering the routing service to replace a simple XSL transformation service with a BPEL-based service that satisfies the new business requirement.

This has important implications for interconnected services. Traditional application development methodologies could not adapt to changing business requirements due to the tightly coupled way in which the business services were woven into the system. By utilizing XML messaging on top of a flexible integration fabric, SOAs provide a basis for resiliency in distributed systems.

## 4 Conclusion

SOAs modularize business services and focus development efforts on the exchange of business information required to support those functions. The increased level of abstraction in a SOA provides enormous benefits to IT organizations: new services are simpler to develop, new automated business processes can be deployed as services, and those processes are amenable to change. We believe these benefits are best achieved by providing programming models and infrastructure frameworks that provide inherent support for SOA enablement and management.

## References

1. Schmidt, Douglas C, Vinoski, Steve: Object Interconnections – Introduction to Distributed Object Computing, SIGS, Vol. 7 Number 1, January 1995
2. OMG, Common Object Request Broker Architecture: Core Specification, Version 3.0.3, March 2004
3. Sun Microsystems Inc.: Java Remote Method Invocation (RMI) Specification 1.5, 2004
4. Waldo, Jim, Wyant, Geoff, Wollrath, Ann, Kendall, Sam: A Note on Distributed Computing, November 1994
5. Sun Microsystems Inc.: JSR 220: Enterprise JavaBeans, Version 3.0, EJB Core Contracts and Requirements, June 25, 2005
6. Service Component Architecture: Building Systems using Service Oriented Architecture, A Joint Whitepaper by BEA, IBM, and Oracle, November 2005.
7. Sun Microsystems Inc.: J2EE Connector Architecture Specification, Version 1.5, Final Release, November 2003
8. Thatte, Satish (Ed.): Business Process Execution Language For Web Services, Version 1.1, May 5, 2003

# Describing the Architecture of Service-Oriented Systems

Vojislav B. Mišić and Michael W. Rennie

Department of Computer Science
University of Manitoba, Winnipeg, Manitoba, Canada
vmisic@cs.umanitoba.ca

**Abstract.** Effective design and implementation of service-based systems requires proper methodology and tool support at all levels. At the architectural level, such support includes a suitable architecture description language (ADL), an example of which is described in this paper. A prototype implementation of the proposed ADL generates Java code skeletons which can be used to build Jini-compliant applications.

## 1 Introduction

The service-based computing paradigm is rapidly gaining acceptance as a viable option for the creation of modern software systems. The service-based paradigm is a flexible approach in which software entities are distinguished through the services they provide. In this manner, software systems may be dynamically configured as a loosely coupled association of entities, some (or most) of which are loaded on demand and unloaded when their services are not needed.

However, advances in several areas are needed before the service-based paradigm can find wider use for mission-critical industry applications. One of the areas in which such advances are needed is the infrastructure to support the basic operations of the service-based paradigm – e.g., along the lines of [1]. Another one is software architecture [2] which has to make use of the service-based paradigm right from the start, rather than applying it later as an afterthought. An important step towards that goal is the development of service-oriented architectural description languages (ADLs), a number of which have been proposed in the last decade [3]. Unfortunately, most of the existing languages have focused on the problems related to static architectures, while the (much more interesting) issues related to dynamism were not as popular. In fact, only a handful of architecture description languages provide support for dynamism, and no language that we know of deals specifically with the requirements of the service-based computing paradigm. The lack of a suitable ADL presents an impediment to further development and wider acceptance of service-based systems and applications. To address this lack, we propose a small XML-compliant language, named SeAL (for Service-oriented Architecture description Language). We also outline a tool to create and validate specifications written in this language and to subsequently generate the skeleton code to implement them, and present some findings from a prototype implementation of such tool.

The paper is organized as follows. In Section 2 we briefly review major concepts of service-based design. Section 3 gives an outline of SeAL, while some notes regarding the prototype tool are given in Section 4. Finally, Section 5 concludes the paper.

## 2 Main concepts of service-based design

In component-based systems, components are the bricks from which systems are built: they are units of packaging as well as units of deployment. A component provides or implements some services for others to use; in order to do that, it requires other services which are provided by other components. Services exist but act merely as plugs and receptacles through which the components interconnect and subsequently interoperate.

Service-based systems are built upon the concept of services as first class entities, which then interoperate to form a dynamic and evolving system. The connections are formed only when needed, they exist only while the interaction takes place, and they can be destroyed afterward. Components are still needed, but only to provide convenient packaging for the services.

This approach has a nuumber of distinct advantages [4]. Alternate service implementations can be substituted as long as they implement the same externally observable behavior. An explicit request for a particular service may be deferred until the most suitable provider is found, and the selection criteria may vay from one invocation to another. Interactions may take place locally or remotely, as the location of the service provider may be irrelevant to the client. (Of course, local services may be preferred on account of performance, security, or other properties.) In either case, services interact through a series of asynchronous messages exchanged between the client and the service provider. resulting in a true message passing paradigm.

As mentioned above, service interactions necessitate the presence of an infrastructure capable of managing them, either as part of the original client application or independently of it. Such an infrastructure may be embedded in the operating system and thus made available to all the applications, or it may be made to run as part of the actual application. (The choice, of course, depends on the facilities offered by the operating system.) A number of such frameworks currently exist, most notably Web Services [5], which are rapidly increasing in popularity despite the fact that they support remote service invocation only.

## 3 The SeAL language

Let us now present the main tenets of the SeAL language, with comments as appropriate.

**Architectures.** A service-based system is described through a series of architectures, which can contain other, nested architectures which may optionally be defined. Architectures can thus be reused when needed, which provides a high level of flexibility.

```
Specification :== Architecture+

Architecture :== Openness "architecture" name "is"
                 "starting-with" Service
                 ["contains" Service]*
                 ["includes" Architecture]*
                 "end-architecture"
```

```
Openness :== "open" | "closed"
```

In case of nested architectures, services from a higher-level architectures have access to all the services from the lower level ones. On the other hand, services from a lower level architecture can access services from the higher level one if and only if (a) the lower level architecture is not declared closed, and (b) such services are explicitly designated as shareable, in the manner that will be outlined below. This provision may seem as a violation of the principle of information hiding, but it actually facilitates reuse, as an architecture can reference other architectures which are specified elsewhere, whilst information hiding is supported at the service level, similar to most object-oriented programming systems.

Each architecture definition must include the definition of a starting service – the service which is to be run when the implementation of the architecture is executed. Other services within the architecture may be loaded at the same time or later, depending on the service definitions and available resources, as will be seen below.

**Services.** Each service within an architecture may optionally be provided (i.e., implemented) by one or more software components. If a service is declared as external, its implementation resides outside of the architecture, as in the case of Web Services, and no implementing component can be specified. Specifications of non-external services may include the implementing component.

Most services are accessible globally, which means that other services, both within the architecture and outside of it, can access them. (Global accessibility is the default.) If the service accessibility is defined as local, only the services within the same architecture can access it. Accessibility is thus an implementation of the principle of information hiding.

```
Service :== Accessibility "service" name "is"
            [["external"] | ["provided-by" Component]]
            ["provides" ServiceMsg]+
            "end-service"

Accessibility :== "local" | "global"
```

**Components.** A component is a packaging unit with which is used to physically implement services. An important property—from the architectural viewpoint—is the component's availability. A *private* component provides its services to clients within the same architecture only; such services are always local. A *protected* component is accessible to services residing outside of the architecture; however, any interaction must be performed through message exchanges only (again, think of Web Services); services provided by a protected component may be either local or global. Finally, a *public* component may have its executable image (e.g., a Java .jar file or equivalent) available to be transferred to the remote host for execution.

```
Component :== Availability "component" name
              "end-component"

Availability :== "private" | "public" | "protected"
```

Access restrictions imposed by the components are mapped onto services. In this manner, a service can have different implementations, some of which are global while the others are protected, and possibly some that are local as well. Note that other services within the same architecture will be able to access all of those services.

The default accessibility level is public; protected takes precedence over public, and private takes precedence over either of them; this model is similar to the one adopted in Java. Since the same service provider component may provide more than one service, the access restrictions specified within the defitinions of those services may differ. In this case, the most restrictive qualifier will be used, eliminating inconsistencies from the processing of architecture definitions.

Note that the outward extension of an architecture depends entirely on the service provider definitions, whereas the inward extension depends on the architecture itself. In other words, an architecture is free to 'close' itself by declaring that it will not seek help from others. Irrespective of the extendibility setting, individual services may be accessible (and their provider components may be available) to the outside world.

It is important to note that a component defined as public does not mean that services it implements are mobile; they are only movable. In other words, it can be sent between clients and executed on it; but it can't be suspended in mid-execution and then transferred to another client in order to resume execution. (Extensions of the SeAL language to support such behavior are the topic of our current work.)

**Service Messages.** Each service is invoked via a specified message. The first part of the message specifies functional information such as the service name and a list of parameters in parentheses; this is rather similar to method signatures. The other, optional parts specify quality of service (QoS) promises as well as the required preconditions. Each QoS promise consists of a property and the associated value; this information may be used to guide the process of service selection. However, in order to fulfill those promises, a component may require certain preconditions to hold.

Two main types of preconditions can be readily identified: resource requirements and service requirements. Resource requirements are analogous to the QoS promises, except that they spell out what are the properties of the operational environment that the client must provide in order for QoS guarantees to be met: for example, available memory, CPU speed, specific version of the operating system, and the like.

```
ServiceMsg :== name "(" ParameterList ")
               ["with" QoSGuarantee]+
               ["at" ResourceReq]*
               ["requiring" RequiredServ]*

ParameterList :== [Datatype [", " Datatype]*]

QoSGuarantee :== Property "of" Value

ResourceReq :== Resource "of" Value
```

These definitions facilitate the use of a QoS specification protocol similar to those proposed for Web Services, such as the WSOL [6]. Additionally, a shared QoS ontology

along the lines of [7] could be defined so as to ensure compatibility among different specifications.

Service requirements, on the other hand, identify a number of other services that are, or may be, needed in order to fulfill the obligations. Furthermore, certain QoS requirements may be specified for those services as well.

```
RequiredServ :== Location Immediacy name
                 ["(" ParameterList ")"]
                 ["with" QoSGuarantee]+

Location :== "local" | "remote"

Immediacy :== "immediate" | "optional"
```

This kind of dependency may be limited to services available locally, i.e., those defined within the architecture as well as those downloaded from other architectures; the default option is to include all services, local and remote ones. If a required service is labeled immediate, it must be made available prior to execution by whatever means available. If it is optional, its acquisition may be deferred because the component is willing to wait for it when needed, or might not even need it at all.

The facilities described above provides an additional selection criterion that allows for finer control of system execution and, consequently, performance. Note that any single service can be provided by more than one component; the QoS guarantees provided by those components may differ, as may be the case with the required preconditions. Thus some clients in need of a service may opt for best performance regardless of its extended requirements, while the more cost-conscious clients may prefer to get service that require fewer or less additional resources.

The infrastructure that manages the architecture will initially load the starting service, as well as its immediate required services (subject to resource limitations, of course); optional services will be loaded when they are actually invoked.

Finally, we note that the list of required services is an optional part of the SeAL language, limited to the definitions of local services – if a service is accessed via a remote host, no guarantees can be given as to the services it may require.

## 4 The implementation

For the implementation of the SeAL language, we have chosen to employ an XML compliant notation. In this manner, we can leverage all the benefits offered by XML, most notably platform independence, ease of processing (as XML parsers are readily available), and simple validation (provided a validating parser is used). To that end, the grammar described above has been mapped to aan XML Schema, which (as is well known) offer better expresiveness and better control over document content than a comparable DTD. A simple example is given below.

```
<architecture documentation="" id="a6"
 name="Louvre" openness="closed">
```

```
    <starting-with>
      <service accessibility="global" documentation=""
       external="false" id="s5"
       name="Greek" provided-by-component="c0 ">
      <component availability="protected" documentation=""
       id="c0" name="Athens" provides-service="s5 "/>
        <service-message documentation="" id="sm1"
         name="welcome">
          <parameter-list documentation="" id="pl2"
           name="Parameters">
            <parameter documentation="" id="p18"
             name="fromWho" value="String"/>
            <parameter documentation="" id="p19"
             name="toWhom" value="String"/>
          </parameter-list>
          <qos-guarantee documentation="" id="q20"
           name="loudness" value="100"/>
          <required-resource documentation="" id="rr21"
           name="silence" value="50"/>
        </service-message>
...
    </starting-with>
...
  </architecture>
```

The next step in implementing SeAL was to build a prototype tool for editing, validating, and code generation; the tool was implemented in Java. A screenshot of the tool's main window is shown in Fig. 1.

As for the code generation, we did consider the possibility of creating the Web Service skeletons; however, this would limit the architecture to external services only. Because of that, we have focuse on the Jini framework [8]. While Jini is not the most recent development, the facilities it offers match the constructs in the SeAL language rather well. For example, the generated source code is grouped into packages based on the architectures specified: each architecture is contained within its own package, which allows us to enforce the openness restrictions. Availability and accessibility can be supported as well, as all of the sub-elements of an architecture in the design are constituents of the corresponding package. Finally, the latest version of the framework does offer the possibility to create Web Services-compatible packages as well.

At present, the source code generated by the tool is not editable from within the tool; however, given the multitude of excellent support tools available, this was not considered as a high priority task. All source that is created is fully Java compliant and provides javadoc recognizable commenting of classes, interfaces and methods.

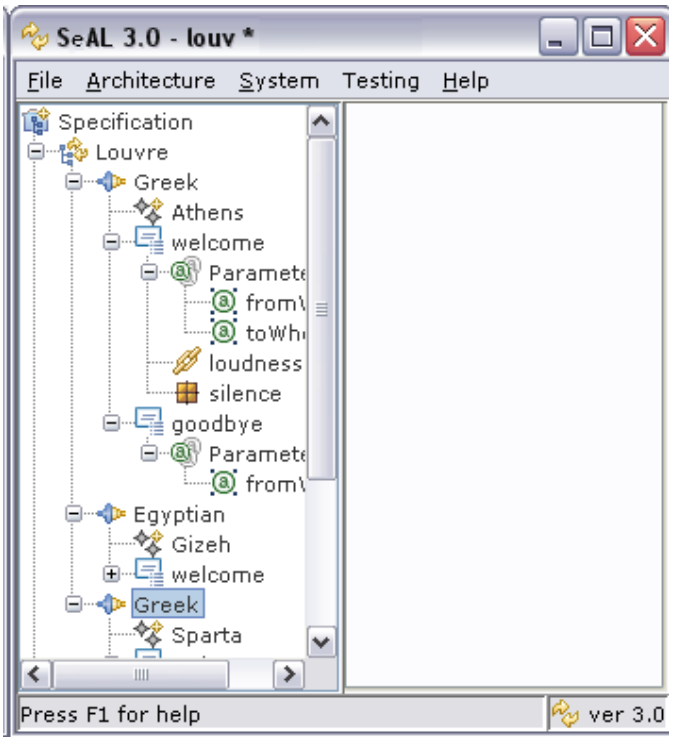


**Fig. 1.** A screen shot of the prototype editing and validation tool.

## 5 Still much to be done

Of course, the work reported here is but the beginning, and much remains to be done. We are currently working in two directions. First, we are considering building a runtime environment for testing the specification in a live setting. Speaking of interoperability, we could make use of one of several available component-centric evironments, such as those baed on the OSGI [9] technology and its successor, Web Services Resource Framework [10]. Code generation could also be improved, and the tool could perhaps be redesigned as an Eclipse [11] plugin.

Second, we are considering migration to a different development tool, most notably ArchStudio 3 [12]; this would also mean that SeAL would have to be modified to become compatible with the xADL 2.0 extensible architecture description language [13].

Moreover, we are looking into the methodologies to specify and design the architecture of service-based systems. Regardless of the current (relative) simplicity of this project, we believe that it offers a promising proof that the design of service-based applications can be undertaken with ease, and that the service-centric way of looking at applications is indeed the way future software applications will be designed and built.

## References


1. Maximilien, E., Singh, M.: A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing **8** (2004) 84–93
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. 2nd edn. The SEI Series in Software Engineering. Addison-Wesley, Reading, MA (2002)
3. Medvidovic, N., Taylor, R.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering **26** (2000) 70–93
4. Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, New York, NY (2005)
5. Fletcher, P., Waterhouse, M., eds.: Web Services Business Strategies and Architectures. Expert Press, Chicago, IL (2002)
6. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web Service Offerings Infrastructure – A Management Infrastructure for XML Web Services. In: Proceedings IEEE/IFIP Network Operations and Management Symposium. Volume 1., Seoul, Korea (2004) 817–830
7. Ankolenkar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In: Proceedings of the First International Semantic Web Conference, Sardinia ,Italy (2002)
8. Sun Microsystems, Inc.: Jini Network Technology. http://www.sun.com/software/jini (2004)
9. OSGi Alliance: Open Services Gateway Initiative. available at `http://www.osgi.org/` (accessed on 10 November 2005)
10. Web Service Resource 1.2 (WS_Resource). Public Review Draft 02, OASIS Open, available at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf` (accessed on 06 October 2005)
11. IBM: The Eclipse Project. http://www.Eclipse.org (2004)
12. ArchStudio 3 Software Architecture-Based Development Environment. Institute for Software Research, University of California, Irvine, Irvine, CA, available at `http://www.isr.uci.edu/projects/archstudio/` (accessed on 10 November 2005)
13. xADL 2.0 Highly-extensible Architecture Description Language for Software and Systems. Institute for Software Research, University of California, Irvine, Irvine, CA, available at `http://www.isr.uci.edu/projects/xarchuci/` (accessed on 10 November 2005)

# Bridging the Gap between Business Processes and existing IT Functionality

Gero Decker

Hasso-Plattner-Institute for Software Systems Engineering at the University of Potsdam, Germany
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
gero.decker@hpi.uni-potsdam.de

**Abstract.** Transforming business processes into IT supported processes can be very challenging. The functionality exposed by existing information systems often does not match the tasks defined in the business process model. Reasons are inappropriate granularity, unsuitable interaction models, and complex interdependency inside the enterprise information systems.
This paper examines existing solutions for retrieving IT supported processes. We then present a list of incompatibilities between business process models and existing IT functionality.
We propose patterns for introducing a process support layer that solves some of the incompatibilities. The effectiveness of this approach is shown by applying it to a real-world example.

## 1 Introduction

A clear trend towards process orientation can be observed in software technology. Business processes move into the center of attention. But how can we use the already existing enterprise information systems for enacting business processes? How much work has to be done to bring a process model and the existing functionality together?

Closely supporting business processes within a company and between business partners is one of the main purposes of service-oriented architectures (SOA). As described in [1] we distinguish between three elements within a SOA: Service providers, service requesters, and a service broker. Providers publish their services to the broker. Requesters can use the broker to find desired services. After having bound to a service, the requester can finally use the service.

There exist a lot of different definitions for the term "service" e.g. in [2], [3], and [4]. In order to avoid ambiguity we will focus on services that either realize intra-enterprise business processes or provide access to functionality of a certain application domain. In [4] these types of services are called process-centric services and basic services, respectively. Furthermore, we will only focus on services that are technically realized as web services. According to the terminology introduced in [5] we will talk about service operations that can be invoked. In

addition, we will distinguish between stateless and stateful services. While stateless services do not preserve conversational state between operation invocations, stateful services do so. However, using a stateless service can affect the state of an underlying system.

When applying typical engineering process models like the Rational Unified Process [6] to the development of a SOA we would first perform a business analysis. This leads to a business process model containing a set of (business) tasks, a description of the control flow, and a high level definition for the data flow between the tasks. We then want to automate the enactment of the business processes according to the given model. An elegant way to realize this is proposed in [4] where we employ a business process management system (BPMS). We have to tell the BPMS which functionality of the enterprise information systems is to be used for performing the individual tasks. Therefore, we take our task specifications and try to match each task to a service registered in the broker.

Since we have carried out the business analysis without having in mind what services are actually available, we will probably run into incompatibilities between the existing services and our tasks. For instance the granularity of the tasks could be different from what is offered by the systems: We could have modeled a business task "source material" while we can only use service operations such as "source material from stock", "select supplier", and "source material from supplier".

Existing solutions for these incompatibility problems will be discussed in the next section, before we suggest introducing a process support layer in section 3. An insight into different patterns for this layer is provided in sections 4 and 5. Some of these patterns are applied to a real-world example taken from [8] in section 6. Finally a conclusion and outlook will be given.

## 2 Existing approaches

As we have already said, tasks within a business process model can often not directly be mapped to existing services. The effectiveness of approaches for solving these incompatibilities can be measured by the solution's initial costs, the costs of changes in the business process model (maintenance costs), the degree of modification of the model, and the number of solvable incompatibilities.

**Approach 1: Changing the business process model.** Changing a business process model for implementation reasons is generally not desirable. When changing the model we might run into the problem that the business analysts and managers do not understand the resulting model any longer. They might not recognize how their business' individuality is reflected within the resulting model. Another problem of this approach occurs when a redesign of the business processes is necessary because of changing business requirements or strategies

later on. Then we probably run into incompatibilities once again, which in turn forces us to change this new model.

Some ERP system vendors go one step further: They propose that a business process model should be designed having in mind what services the system offers. Thus, a distinction between analysis and realization phase is eliminated. This forces the business analyst to exactly adopt the terminology and granularity that is predefined by the given systems. However, since older ERP systems have been forcing companies to use this terminology for a long time now, a lot of the analysts got used to it.

**Approach 2: Changing the information system.** This option can sometimes be found in the industry, too. Engineers of ERP system providers have to be paid for implementing new tailor-made services within the existing systems. This solution takes time and can be very costly. Maintenance of the business process model also becomes costly: Every time the model is changed, it is likely that new tailor-made services are needed in order to avoid the newly arising incompatibilities.

This approach is only feasible for a company if it carries out a pilot project where it closely collaborates with the ERP system provider.

**Approach 3: Maintaining two different process models.** This option leaves the initial business process model unchanged. However, it has the disadvantage that we now have to deal with two different models. They have to be constantly kept in sync. It is not sufficient to derive the second model once, but every time the business process model is changed we also have to adapt the second model.

The strategies proposed in [7] only allow strict functional decomposition for realizing business process tasks. Henkel et al. [8] propose more transformations but still only solve a limited number of incompatibilities. Thus, they conclude that their approach is not sufficient for real-world scenarios.

## 3 The Process Support Layer

We now propose a fourth option of which we think that is more effective than the three other ones in some scenarios.

Like in the first option, we directly enact the business processes. However, we introduce an architectural layer between the BPMS and the existing information systems. This process support layer copes with the incompatibilities and makes them transparent to the BPMS. That way, we can leave the business process model unchanged.

Since every task within the business process model has to be mapped to a service task we build supporting services that cope with the different incompatibilities and offer exactly the operations we need.

Based on the transformation list provided in [8] and other possible incompatibilities, we established a set of patterns that point out solutions to the given

challenges. We describe how supporting services can internally look like for each pattern. Hence, a software engineer does not need to reinvent the wheel for each incompatibility but can consult the patterns to quickly come to a solution.

## 4 Process Support Layer Patterns

| **Pattern name** | **Problem** | **Solution** |
|---|---|---|
| Composition | The existing business functions are too fine-grained. | We introduce a composite service aggregating the existing functions. |
| Decomposition | The existing business function is too coarse-grained. | We introduce a stateful service that calls the existing function as soon as enough data has been gathered. |
| Technical Switch | The business functions offer different technical solutions for the same task. | We introduce a service that decides what function is actually called. |
| Bulk Service | The business function processes only one item at a time while we need to process several ones. | We introduce a service that performs several calls within a transaction. |
| Blocking Send/Receive | The existing business function is asynchronous while we need to perform a synchronous call. | We introduce a service that performs an asynchronous call and waits for the corresponding result. |
| Non-blocking Send | The existing business function is synchronous but we want to perform an asynchronous call. | We introduce a service that performs the call while the business process can continue. |
| Non-blocking Send/Receive | The existing business function is synchronous but we want to proceed while the service is working. | We introduce a service that performs the call while the business process can continue. The result is passed to the business process. |
| Sequentializing | The existing business functions have to be called sequentially while we want to call them concurrently. | We introduce a stateful service that waits for the right operation calls to be performed and calls the existing functions in the appropriate order. |
| Reordering | The existing business functions have to be called in an other order than we want to. | We introduce a stateful service that calls the existing functions in the right order as soon as enough data is available. |

**Table 1.** Process Support Layer Patterns

Granularity is the amount of computation performed by a function. Incompatibilities between business process models and existing business functions arise if the functions' granularity does not match the business tasks' granularity. The patterns named in the first four rows of table 1 help to bridge the gap between different levels of granularity.

The patterns for interaction model problems enlisted in the middle part of table 1 are strongly inspired by the Service Interaction Patterns presented in [11]. The authors categorize possible scenarios of service interaction by the number of parties involved (bilateral vs. multilateral interactions) and the maximum number of exchanges (single-transmission vs. multi-transmission interactions).

We have taken a closer look at the bilateral, single-transmission interaction patterns and we have examined what happens if the two parties involved support different interaction patterns.

In order to compose different business functions to form a business process these functions should ideally be independent from each other. However, the business functions of existing information systems often have to be called in a specific order. Two incompatibilities can be solved by applying the patterns named in the last two rows of table 1.

The following section will cover the Decomposition pattern. A more detailed description of this and the other patterns can be found in [12].

## 5 Decomposition

**Problem:** The existing business function is too coarse-grained.

**Example:** The existing systems offer the business function "check invoice" while the business process model contains the tasks "check for duplicates" and "check invoice completeness" and "check invoice/order correspondence". All the functionality specified in these three tasks is incorporated in "check invoice".

**Solution:** A stateful service that offers all the desired operations is introduced. This service performs one call on the existing business function as soon as enough data has been gathered.

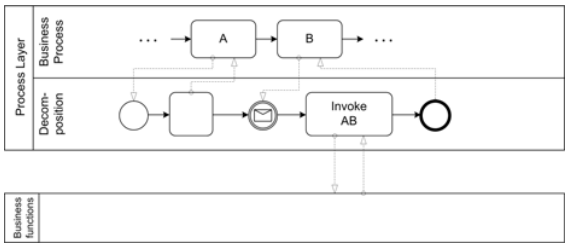


**Fig. 1.** Decomposition pattern

Figure 1 shows how an existing business function AB is decomposed into A and B. This figure uses the business process modeling notation [10].

**Applicability:** When applying this pattern you have to carefully consider your data flow. There are examples where the proposed solution does not work.

Imagine introducing a third task C between A and B. As regards the data flow, A influences C and C influences B. This problem might sometimes be solved

by calling AB twice. E.g. if both A and B are read-only tasks that do not change the state of the underlying information system. In other cases it is impossible to call AB twice. Let us consider the example where A is "generate offers for customer", B is "filter offers", and C is "send offers to customer". This case can neither be solved by applying Decomposition described above nor by calling the business function twice.

## 6 Example Case

In [8] we can find a simplified real-world business process model. In this model an order is issued by the customer. The order is then confirmed, before it is processed and the shipment is planned. Finally the shipment advice is sent to the customer. Figure 2 illustrates this process.

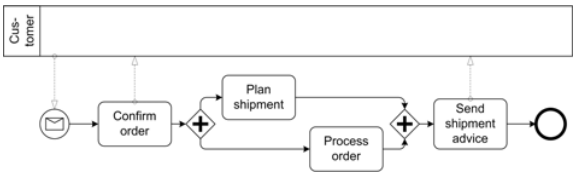


**Fig. 2.** Business process model from [8]

The following restrictions are imposed by the existing services:

1. An existing service triggers both logistics planning and order processing.
2. A notification is returned after this asynchronous service operation has completed its work.
3. An order confirmation can be sent to the customer as soon as the order is received by the service.
4. The confirmation is sent as a HTTP message or a FTP file transfer depending on the customer's service ability. We assume that for both alternatives a service is available.

All of these limitations can be tackled by introducing a process support layer. The supporting services can be designed by using the patterns enlisted above.

Restriction 1 is a granularity problem that can be solved by using Decomposition. Since Decomposition expects sequential calls of the operations we have to introduce Sequentializing, too. Restriction 2 causes an interaction model incompatibility because the task within the business process model implies a synchronous call. Here we apply Blocking Send/Receive. Constraint 3 poses the problem that the confirm order task can only be executed after the logistics planning and order processing have been completed. Thus, Reordering is used. Finally, Technical Switch provides a solution for restriction 4. Figure 3 illustrates the dynamic structure of our solution where the patterns proposed are employed.

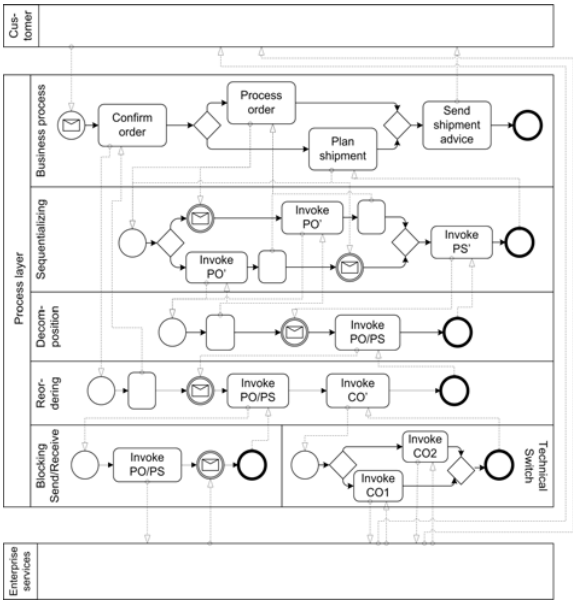


**Fig. 3.** Dynamic structure of the supporting services

In [8] it is stated that a lossless realization of the given business process is not possible because Reordering and Decomposition are not allowed. A refactoring of the business process model, which we discussed in section 2, is proposed as the only valid solution.

As we have shown, this unsatisfying suggestion can be overcome by introducing a process support layer and by applying the patterns.

## 7 Conclusion and Outlook

We have presented a new approach for solving incompatibilities between business process models and existing business functions. As depicted in table 2 our approach (appr. 4) is more effective than the existing ones in certain scenarios.

| | **Initial costs** | **Maintenance costs** | **Business model modification** | **# solvable incompatibilities** |
|---|---|---|---|---|
| **Appr. 1** | medium | medium | high | high |
| **Appr. 2** | high | high | very low | high |
| **Appr. 3** | medium | medium | very low | medium |
| **Appr. 4** | **medium** | **medium** | **very low** | **high** |

**Table 2.** Comparison between our approach and the existing ones

We have proposed a number of patterns which help to develop supporting services and therefore to reduce the realization costs. This pattern list can surely be extended and more research is needed to identify the most helpful patterns.

We have not discussed impacts of our approach to performance measuring. Probably the measuring results are jeopardized because we sometimes completely reorder the tasks in the process support layer. This might also result in strange behavior when it comes to logging: The order of log entries might not directly correspond to the business process model any longer, which in turn might have an impact on the compliance with legal requirements.

Furthermore, we have not addressed the general problems of stateful services. Since we enact process instances within the supporting services we have to ensure that these instances terminate. So what do we do if an instance dies?

Some incompatibility problems cannot be solved using our approach. E.g. there are business functions that affect several application silos. This screws a clear process design where each task is to affect only one silo. In this case it is up to the ERP system providers to refactor their products in order to provide highly reusable business functions. If application silos are separated cleanly and functions are comprehensibly exposed, faster and more efficient development of easily maintainable enterprise applications will probably be possible.

## References

1. IBM: Web Services architecture overview. (2000) http://www-128.ibm.com/developerworks/webservices/library/w-ovr/
2. Fremantle, P., Weerawarana, S., Khalaf, R.: Enterprise Services. Communications of the ACM, ACM Press New York, NY, USA (2002)
3. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. (2002) citeseer.ist.psu.edu/foster02physiology.html
4. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA. Prentice Hall (2004)
5. W3C: Web Services architecture. (2004) http://www.w3.org/TR/ws-arch/
6. Kruchten, P.: The Rational Unified Process: An Introduction, Second Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)
7. Dijkman, R. M., Quartel, D. A. C., Ferreira Pires, L., van Sinderen, M. J.: A Rigorous Approach to Relate Enterprise and Computational Viewpoints. In: Proceedings of the 8th IEEE Enterprise Distributed Object Computing (EDOC) Conference, Monterey, CA, USA, pp. 187-200 (2004)
8. Henkel, M., Zdravkovic, J., Johannesson, P.: Service-based Processes - Design for Business and Technology. (2004)
9. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. Oxford University Press (1977)
10. BPMI: Business Process Modeling Notation (BPMN) Specification, Version 1.0. (2004) http://www.bpmn.org/Documents/BPMN V1-0 May 3 2004.pdf
11. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns: Towards a Reference Framework for Service-Based Business Process Interconnection. (2005) http://sky.fit.qut.edu.au/~dumas/ServiceInteractionPatterns.pdf
12. Decker, G.: Bridging the Gap between Business Processes and existing IT Functionality. (2005) http://myhpi.de/~gdecker/BridgingTheGap.pdf

# Designing Reusable Services: An Experiential Perspective for the Securities Trading Domain

Abdelkarim Erradi [1], Naveen Kulkarni[2], Sriram Anand[2], Srinivas Padmanabhuni[2]

[1] School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
aerradi@cse.unsw.edu.au
[2] Software Engineering and Technology Labs
Infosys Technologies Ltd, Bangalore, India
{Naveen_Kulkarni, Sriram_Anand, Srinivas_P}@infosys.com

**Abstract.** Service-oriented Computing (SoC) is an approach for building distributed systems that deliver application functionality as a set of self-contained business-aligned services with well-defined and discoverable interfaces. This paper presents our experiences in designing services for the securities trading domain. Best practices and lessons learned during this exercise are also discussed. Further, it discusses a range of techniques and guidelines for systematically identifying services, designing them and deciding service granularity.

## 1. Introduction

The increasing move towards end to end automation of business processes has fuelled interest in Service-oriented Computing (SoC), which is focused on the integration of heterogeneous, autonomous software systems with virtualization and "black-box" reuse as the central theme. SoC decomposes a system in terms of services and focuses on loosely coupled message based interactions [6], and forces separation of service interface description, implementation, binding and declarative policies and service level agreements (SLAs) governing service interactions.

An effective approach for modeling and designing services is crucial for achieving the full benefits of SoC with the optimal level of service granularity being an important consideration. In this paper, we present the set of design principles and processes for identifying, designing and layering services in a repeatable and non-arbitrary fashion. This has been derived from an elaborate SoC example involving the Securities Trading domain. The rationale behind design decisions is captured and the lessons learned are reported. The resulting prototype implementation of the case study has been accepted for presentation as a demo at ICSOC'05.

The rest of the paper is organized as follows. In Section 2 we provide an overview of the securities domain focusing on the pain points inherent in this area. Subsequently, in Section 3 we briefly discuss our suggested service-based decomposition framework. Section 4 details the suggested service design for our case study. Section 5 presents the lessons learned and the key service design considerations. Section 6 briefly discusses related work and reviews their limitations. The last section concludes the paper and provides some directions for future work.

## 2. Background and problem area

In this paper, we concentrate on the area of securities trading that relates to the order capture, processing and fulfillment of equities trading. The captured order needs to be priced and validated for correctness. Subsequently, the order is sent to an exchange for fulfillment. The exchange may fulfill the order in batches and send the acknowledgement back to the order management system. Subsequently, trades must be allocated to the right order, matched and the information sent to the custodian. Finally, the funds are transferred and the data is sent to the Depository for recordkeeping.

The typical issues that are encountered by business and IT groups in existing IT architectures that enable securities trading processes can be enumerated as below:

**Heterogeneous IT portfolio:** The typical IT portfolio of large brokerages is heterogeneous and contains multiple systems that are usually integrated using proprietary and brittle point to point connections that impact flexibility.

**Redundant and overlapping functionality:** Most brokerages offer multiple financial instruments to their customers with business process and IT portfolios for these business offerings having been developed independently in silos. This leads to a redundancy in certain processes and IT systems leading to cost overheads and increased time to market. A specific example may be the use of individual pricing engines along with individual market data servers for multiple trading instruments.

**Inflexible and costly legacy applications portfolio:** In many cases, a large chunk of mission critical functionality resides on legacy systems with high cost of ownership including costs of maintenance, operation and upgrade of both software and hardware. Legacy platforms are typically inflexible due to the proliferation of unstructured code and the lack of documentation of key modules.

In order to address the above mentioned issues, we propose an enterprise level incorporation of SoC to yield a future-proof SOA. We use a structured architectural methodology termed as Service Oriented Architecture Framework (SOAF) [2], to systematically review the architectural pain points and develop an enterprise wide SOA. The main business drivers for adopting service-orientation for our case-study are: to accelerate the securities trade processing from T+3 processing towards Straight Through Processing (STP), and to make the securities trading accessible from various channels like Web and mobile devices.

## 3. Service Oriented Decomposition Process

Service-based decomposition is an iterative process for arriving at an optimal services composition. The aim is to first establish clear and well-defined boundaries between collaborating systems, followed by reduction of interdependencies and limiting of interactions to well-defined points. The key tasks in the process include identification of services along with appropriate layering of services.

As shown in Figure 1, for service identification we advocate a hybrid approach combining top-down domain decomposition along with bottom-up application portfolio analysis. This yields a list of candidate services that further need to be rationalized and consolidated. The top-down analysis of a business may be decomposed into products, channels, business processes, business activities, use cases etc. The business activities are often good candidates for business services. For

example, the activity of obtaining a price for a specific security during an equity trading business process may be a logical candidate service. On the other hand, a broker could offer equity trading as a product which requires instantiating order placement and settlement processes, whose activities could be realized by services harvested from functionalities embedded in existing applications. The harvesting can be facilitated by reverse-engineering techniques and tools to extract data and control flow graphs that provide different views of abstraction of operational systems.

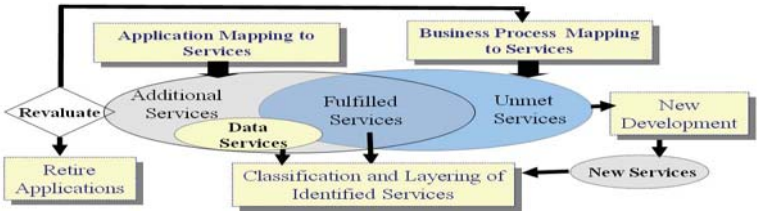


**Figure 1: Service identification framework**

Service identification also covers identifying reusable infrastructure services that may be leveraged to support business services like security and provisioning services.

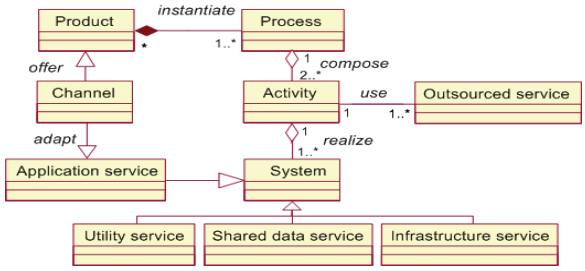


**Figure 2: Service conceptualization Meta-model**

Figure 2 shows a way to guide service based decomposition activities: (1) identification of candidate services representing communication points between the parties involved, followed by (2) capturing and describing a black-box view of the service representing the externally observable behavior. An illustration of service-based decomposition of the Securities Trading application is depicted in Figure 3. During the service identification the primary view point should be towards achieving a common business goal through a single service. The business processes usually are modeled to achieve a single goal and hence would provide a natural boundary. For example a Trade Settlement service would aggregate various correlated activities like allocation matching, trade billing (commission, tax, fees etc) to achieve the goal of trade settlement.

The identified services can be classified and grouped in a variety of ways. The services can be classified according to their scope into cross-business services, cross LOBs/channels services, and LOB/channel specific services. The classification can also be based on their degree of reuse such as core enterprise services used by all (like a Customer Information Service), common services, or services unique to a

specific application. The service classification activity is crucial to guide the non-functional aspects of services design, for example core and common services need to be designed and deployed with more emphasis on scalability and high availability.

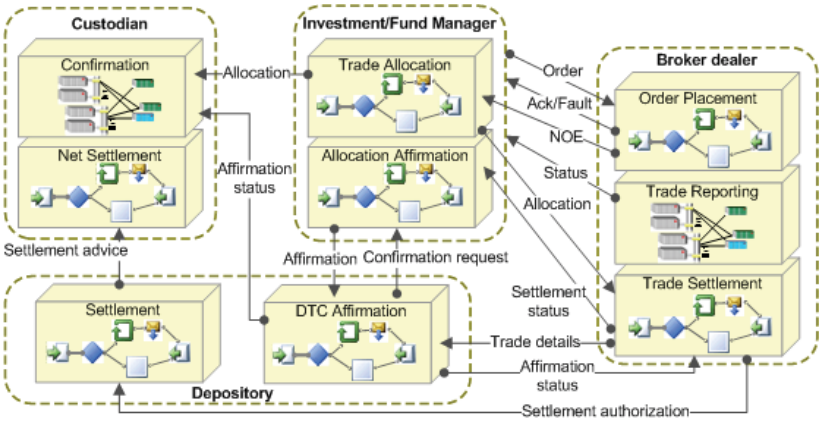


**Figure 3: High-level view of key Securities Trading services and their choreography**

## 4. Service Design

This Section briefly presents key service design principles. Then it discusses the main service design decisions for our Securities Trading case study and their rationale

### 4.1. Service design principles

The service design should take into account the basic principle of high cohesion and low coupling among services [5]. This ensures that resulting services are self-contained, replaceable and reusable. **Service Cohesion** refers to the strength of functional/semantic relatedness of activities carried out by a service to realize a business transaction [5]. High cohesion ensures that a service represents a single abstraction and exposed interface elements are closely related to one another. **Service Coupling** refers to the extent to which a service is inter-related with other services, in other words it measures the degree of isolation of one service from changes that happen to another [1]. The objective is to minimize coupling through encapsulation and self-contained services to enable rapid change and to minimize the impact of change. Low coupling can be achieved by reducing the number of connections between services, eliminating unnecessary relationships between them, and by reducing excessive interactions between services [5].

Another key service design principle is that of metadata based stateless service design, meaning that exchanged messages should be self-contained with sufficient information and metadata (like links to persisted data) to allow the destination service to establish the message context [6]. Sound interface design has to anticipate and meet the current and future needs of varied clients using the service in different contexts and different functional and QoS expectations. The service interface should capture and describe externally observable service behavior hiding the

implementation details. This ensures that changes to the implementation are localized and minimize subsequent interface changes.

Optimal service granularity is crucial in ensuring maximum reuse in SoC. If the service is too coarse-grained, the size of the exchanged messages grows and sometimes might carry more data than needed. On the other hand if the service is too fine grained, multiple round trips to the service may be required to get the full functionality. Usually a balance is established, depending upon the level of abstraction, likelihood of change, complexity of the service, and the desired level of cohesion and coupling. A tradeoff needs to be made while taking into account non-functional requirements particularly performance. During service design, reusability can be maximized by using generalized service schema design, where the variations of the service behavior can be captured simply by supplying varying message instances conforming to a subset of a super-schema defined by the service schema.

### 4.2. Service design tasks

Designing service-oriented applications involves a variety of tasks that may be enumerated as below

- Specifying the structure/data model of exchanged messages using a schema definition language such as XML Schema
- Defining the service interface covering the incoming and outgoing messages that are consumed or produced by the service as well as supported Message Exchange Patterns (MEPs), like one-way/notification, request-response.
- Modeling of supported conversations between services by defining the order in which messages can be sent and received
- Specifying the service policy to advertise supported protocols, the constraints on the content of messages and QoS features, like security, manageability assertions, etc.
- Defining the service contract: “terms and conditions” of service usage covering syntactic, logical and semantic constraints governing the service usage.

### 4.3. Services for the Securities Case Study

The suggested service design (as shown in Figure 4) for our Securities Trading case study aims to address the key pain-points discussed in Section 2. It also facilitates the move to STP through increased automation and integration of business processes. The suggested services facilitate real-time communication across the systems belonging to different stake holders. Having a variety of services that fulfill every aspect of the business facilitates automation of many manual activities and opens up greater collaboration opportunities with external partners. The process based services in Figure 4 such as Order placement, Trade reporting, and Trade settlement are provided by the broker to the external partners such as the Investment Manager. For example, the Order placement process would capture the order, validate it and route it to an appropriate order execution service that finally sends the notice of execution (NOE) report once the trade is completed. The NOE would be directly sent to the Investment Manager to trigger the Allocation Matching process which would further call additional services to complete the settlement process. Our design considers four types of services:

- Process services represent workflows that the Broker uses to deliver products offerings, like Equity Trading, through various channels like the Web, telephony or

direct access. Process services, like Order Placement, expose access points that allow business partners to participate in the process. Process services also automate the information flow across disparate systems and eliminate duplicate data entry, manual data transfer and redundant data collection.

- Application services represent business activities that are useful across business units. For example, services like the Securities Pricing service is required across multiple business lines such as equity trading, fixed income trading, asset management, mutual fund trading etc. Application services provide shared and consolidated functional services to reduce/eliminate redundant/overlapping implementations.
- Shared data services map multiple schemas from different data sources to a single schema which is presented to collaborating applications. They provide the ability to unify and hide differences in the way key business entities are represented within the organization or between different business partners. Shared data services, like a Customer service, can expose aggregated entities from specific data sources to reconcile inconsistent data representations and minimize the impact of change.
- Infrastructure services provide shared functions for other services, such as authentication, authorization, encryption, logging, etc. Often infrastructure services can be acquired, like an LDAP directory service, rather than built in-house.

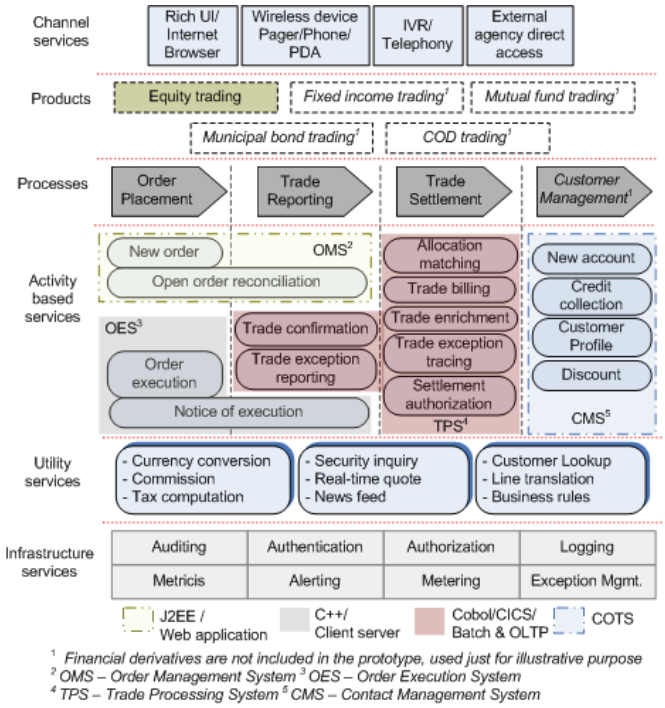


**Figure 4: Equity trading key services from the Broker viewpoint**

## 5. Discussion and lessons learned

This Section discussed the key lessons learned from the Securities Trading case study. Further, key design considerations per service types are briefly presented.

### 5.1. Key lessons learned

While the SoC approach strongly reinforces well-established software design principles such as encapsulation, modularization, and separation of concerns, it also adds additional dimensions such as service choreography, scalable service mediation, and service governance. Our study highlights the following:

- Business process centered top-down identification of shared business services can lead to business aligned service design.
- An enterprise wide common information model (CIM), also known as Canonical Schema, is important to support the consistent representation of key business entities and to reduce syntactic and semantic mapping overheads between services. Standards like STPML [7] for the securities industry should be leveraged.
- Moving to SoC requires more than just a simple change of programming practices, rather a paradigm shift and mindset change is required to switch from RPC-based/object-based architecture to a loosely-coupled, message-focused and service-oriented architecture. A true SoC is realized when applications are built as self-contained, autonomous business services that interact by exchanging messages that adhere to specified contracts
- When service-enabling Mainframe CICS applications, it would be wise to expose one service per screen flow, and avoid translating all transactions to services. This involves identifying the required screens navigation to achieve key capabilities of the application, like CustomerCreation for instance, and then exposing the entire screen flow as a service.
- To ease service discovery and reuse, there is a need for clear service naming guidelines and a services metadata management repository to support governance and easy identification of services based on business function.

### 5.2. Design considerations per service type

For process services design the focus should be on the ease of modification and customization as these services are subject to higher change frequency. Hence, they should declaratively capture only the routing logic to manage the data and control flow between activity services. Further, complex business rules should be abstracted and externalized from processes so that they can be managed by a dedicated rules engine. Further, robust exception handling/compensation design is required.

Application services can have a verb-focused design by exposing key verbs as service methods, which unfortunately require RPC like behavior and sometimes might reveal the internal state of the service. We advocate a message-centric design to allow message content-driven service behavior and generalized service interface that can be used and composed in various applications. Command design pattern is used where the service performs dynamic content-based routing to direct the received messages to the appropriate implementation. This practice is acceptable when the resulting service contract is coherent and deals with closely-related business concepts. For example a generic Securities Price Lookup service could be provided to retrieve

the price from various stock exchanges using content-based routing. Services need to be idempotent so that requests arriving multiple times are only processed once.

Shared data services uses noun-based design and usually expose CRUD interfaces representing simple atomic operations on an entity.

Infrastructure services are usually acquired and act on messages depending on the message context like the channel through which the message has arrived.

## 6. Related Work

Business Applications to Legacy Systems (BALES) methodology is proposed in [4] to support Web-Services development using "objectified" legacy data and functionality to build business applications. However, BALES is OO-focused and yields fine-grained interfaces that are hard to map to coarse grained business processes. Papazoglou et al. [5] describe a design methodology for Web services and business processes. The methodology provides service design guidelines for Web service interfaces and service flow models that maximize cohesion and minimize coupling. Feuerlicht et al [3] focus on the design of domain-specific service interfaces, like the travel domain. However the authors advocate an RPC-based view, while our approach takes a message-centric view. Our approach is more consistent with the latest SoC development best practices and WS-I recommendations [6, 8].

## 7. Conclusion and Future Work

Service-orientation is gaining momentum as a promising approach to deliver increased reusability, flexibility and responsiveness to change. However, the practical design of services requires sound engineering principles. The main contribution of this paper is a service-enablement example in the securities trading domain showing service design best practices and guidelines and highlighting of the challenges therein. Future work will focus on empirical studies of how the level of service granularity affects cohesion and coupling. Further, an in-depth comparison between various service interaction styles, such as REST, MEST [6] and RPC, is highly needed.

## References

[1] Briand, L. C., Daly, J. W. and Wüst, J. 1999, 'A Unified Framework for Coupling Measurement in Object-Oriented Systems', *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91-121.

[2] Erradi, A., Anand, S. and Kulkarni, N. 2006, 'SOAF: An Architectural Framework for Service Definition and Realization', Submitted to ICSE'06, Shanghai, China.

[3] Feuerlicht, G. and Meesathit, S. 2004, 'Design framework for interoperable service interfaces', in *ICSOC'04*, New York, NY, USA, ACM, pp. 299-307.

[4] Heuvel, W.-J. v. d., Hillegersberg, J. v. and Papazoglou, M. P. 2002, 'A methodology to support web-services development using legacy systems', in *IFIP Working Conference on Engineering Information Systems in the Internet Context*, Kanazawa, Japan, pp. 81-103.

[5] Papazoglou, M. P. and Yang, J. 2002, 'Design Methodology for Web Services and Business Processes', in *Proceedings of the Third International Workshop on Technologies for E-Services (TES'02)*, vol. 2444, Springer, Hong Kong, China, pp. 54-64.

[6] Parastatidis, S. and Webber, J. 2005, 'Realising Service Oriented Architectures Using Web Services', in *Service Oriented Computing, MIT Press (chapter obtained from the authors)*.

[7] Straight Through Processing Markup Language (STPML) 2005, http://www.stpml.org

[8] Vinoski, S. 2005, 'RPC Under Fire', *Internet Computing, IEEE*, vol. 9, no. 5, pp. 93-95.

# Towards Style-Oriented SOA Design

Chen Wu, Elizabeth Chang, Vidyasagar Potdar

School of Information Systems, Curtin University of Technology, Perth, Western-Australia, 6845, Australia

{Chen.Wu, ChangE, PotdarV}@cbs.curtin.edu.au

**Abstract.** Architecting service-oriented application is a complex design activity. It involves making trade-offs among a number of interdependent design choices, which are drawn from a range of concerns by various software stakeholders. In order to achieve reliable and efficient SOA design, we believe a rigorous study of architectural style is important. Hence this paper aims at providing a formative survey of existing web services architecture styles extracted both from the academic research projects and industry practices.

## 1 Introduction

According to the web services adoption survey conducted in [3], quality requirements such as system scalability, reliability, and performance have become the second most important criteria for a company to choose web services solutions. Recent research [4] indicates that most of these quality requirements can be heavily influenced by the software architecture (SA) design. Hence for the purpose of architecting better SOA applications fulfilling particular business requirements, this paper provides a literature review of common architectural styles for distributed web services applications based on a classification scheme proposed in existing agent research community. As a result, the architect can leverage existing architectural styles to design web services applications against specific system requirements and resources.

## 2 WS-Architectural Styles

It is well recognized that multi-agent systems can form the fundamental building blocks for distributed software systems, even if the software systems do not require any agent-like behaviors [7]. The classification scheme of architectural styles in this paper is based on earlier work from agent research. It is worth noting that the multi-agent classification scheme used here does not necessarily suggest the architecture components are all agents. We simply apply the middle-agent taxonomy to categorize existing web services architectural styles. One of the most important components in existing multi-agent architecture is the middle-agent [6, 9], which mediates between

the requesters and providers across the Internet. Authors in [9] presented a comprehensive taxonomy for middle-agent in the context of multi-agent systems. In this taxonomy two broad types of middle-agent (*Matchmaker* and *Broker*) are identified. When thinking of introducing middle-agent classification into web services architecture styles, we find that contemporary web services architecture can be grouped into three basic categories: *Matchmaker Style*, *Broker Style*, and *Peer-to-Peer Style*. Moreover, we believe web service lifecycle – *Discovery*, *Execution*, and *Composition* – can be used to scatter these basic styles into variant sub-styles. This is shown in table 1, where each sub-style (e.g. BM) is applicable to some particular phases of web services lifecycle.

**Table 1.** Styles scattered within WS-lifecycle

| | **WS-Discovery** | **WS-Execution** | **WS-Composition** |
|---|---|---|---|
| **Matchmaker** | BM, LM | - | - |
| **Broker** | BB | BB, LB | LB |
| **Peer-to-Peer** | P2PD | P2PE | P2PC-S/M/H |

### 2.1 Basic Matchmaker Style (BM)

The most classical web services architecture is based on matchmaker style, where a matchmaker component acts as a middle agent that stores capabilities advertisements to discover the providers for requesters [6]. Corresponding to table 1, matchmaker does not concern how the services executed, how the services are composed to form service processes. Based on the basic matchmaker style, derived are two matchmaker style variants.

#### 2.1.1 Layered-Matchmaker Style (LM)

Basic matchmaker only allows capability information to be advertised, another important type of information – what the agent community terms as 'preference' – is lacking. For instance, existing UDDI standard lacks the ability to discover and select the most appropriate web services based on non-functional requirements – e.g. 'QoS' – of web Services. To address this open issue, some research ([9], [10], and [11]) adds an additional architectural layer between the service requester/provider and the

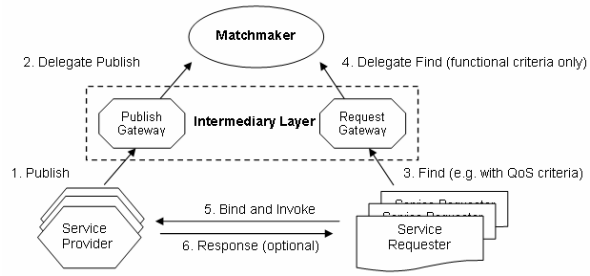


**Fig 1. Layered Matchmaker Style**

matchmaker. Authors in [10] proposed a QoS capable web service architecture, where the extra QoS layer collects QoS data from providers, makes the decision of selecting appropriate providers, and negotiates with selected providers to ensure QoS commitments. The proposed matchmaker – a dynamic service selection engine, which provides QoS-based service selection – is the major work of [9]. Similarly, the architecture offered in [11] allows improved services selection by extended WSDL service interfaces, which in turn can be processed by the additional layer of the UDDI registry. All the above work, in an implicitly or explicitly manner, adds an intermediary layer to the existing matchmaker style for the purpose of augmenting the standard UDDI. We use Layered-Matchmaker style to portray such architectural style depicted in Figure 1. The publish/request gateway forms an intermediary layer that provides additional functionalities (e.g. QoS selection) that is absent in traditional UDDI matchmaker registry. Some extra transaction data is essential at the intermediary layer to augment the standard service discovery and selection process.

### 2.1.2 Federated-Matchmaker Style (FM)

One critical problem of basic matchmaker style is its poor scalability since the centralized matchmaker – UDDI registry – might become the bottleneck and single-point-failure as the number of service requesters/providers increase ([12], [13], and [15]). One may argue that UDDI V3 specification already offers the replication scheme to facilitate collaboration among UDDI nodes scattered in the UDDI clouds. Nevertheless, related research indicates such solution is not feasible. Firstly, it needs complex replication contracts between involved registry providers as well as manual system administration for each registry. Moreover, such replication approach causes extra problems such as expensive data replication, unnecessary global service querying, etc [21]. Therefore, while technically possible, practically replication between UDDI registries does not occur [27].

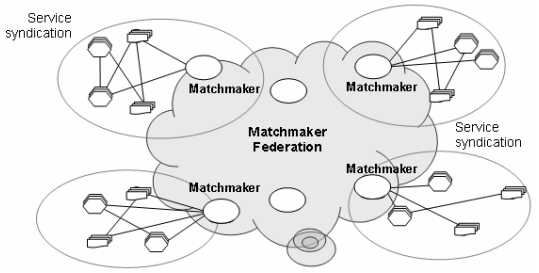


**Fig 2. Federated-Matchmaker Style**

While authors in [14] stated that 'replication was chosen in UDDI because creating a scalable model for distribution of data is inherently difficult', recent researchers attempt to tackle such distribution issue by introducing a *federated matchmaker style* ([12], [14], and [27]). Research in [14] presents a scalable, high performance environment for federated web service publication and discovery among multiple match-

maker registries. Authors in [27] built the matchmaker UDDI federation upon peer-to-peer infrastructure – Edutella P2P overlay. A federation of UDDI-enabled peer registries that operate in a decentralized manner is also provided in [12], where the authors envision the federated matchmaker architecture is able to: 1) support a decentralized service publication and discovery; 2) maintain the same publish/discovery mechanism that existing UDDI has. As shown in Figure 2, service requesters and providers, for some reason – e.g. proximity or functionality, can be syndicated under local matchmakers, which in turn form the matchmaker registry federation in a decentralized way (may or may not be based on peer-to-peer networks). Each local matchmaker is responsible for discovering the immediate services registered within the local syndication. If the requested services cannot be found locally, the registry will form the global query delegated to other registries via the federation network.

### 2.2 Broker Style

Broker style is widely used in distributed information systems such as multi-agent systems and distributed databases. It is a straightforward way to achieve loose-coupling between client and server, and hence fulfilling the requirements of reusability and maintainability. The major difference between a broker and a matchmaker is that the broker involves the execution between requester (client) and provider (server). [1] defines broker architectural pattern (style) as "structures the distributed software systems with decoupled components that interact by remote service invocation. It is responsible for coordinating communications". A broker architectural style includes six components [1]. The most significant component is the *broker* component, which distributes client requests to the responsible server components and returns corresponding results or exception information.

#### 3.2.1 Basic Broker Style (BB)

Besides service discovery and selection, which are already provided by matchmaker, the broker focuses on mediating the interaction between service providers and requesters. Hence the direct peer-to-peer communication in matchmaker style is replaced by indirect message – request, response, and exceptions – delegation offered by the broker in the broker style, where the interaction coupling between provider and requester is removed as indicated in Figure 3. Instead, the message exchange provided by the broker facilitates the interactions. Authors in [17] describe such broker as an 'SOA Fabric' – a central message environment that hides the complexity of reliable message exchanges and other interaction issues from service providers and requesters. Among other interaction issues, the heterogeneity is the most well-known problem and hence received much research momentum [16, 18]. Authors in [16] provided a detailed analysis about broker's architecture requirement, which indicates broker should have powerful reasoning capability in order to homogenize the heterogeneities in different web services. To deal with the heterogeneity, a concrete broker-based architecture style is presented in [18], which employs the extended WSDL files to solve the heterogeneous conflicts.

### 2.3.2 Layered-Broker Style (LB)

It is true that the broker style reduces the complexity involved in developing both service providers and requesters as it makes distribution transparent to the developers [1]. However, the 'hidden' complexity does not mean the complexity is removed or does not exist at all. All the complications now go into the broker itself – the broker needs to handle all the complex problems which used to be handled by providers and requesters. This undoubtedly raises the difficulty to build an 'omnipotent' broker that is capable of handling complicated tasks – discovery, execution, and compose, especially as the number of providers/requesters increases exponentially across the Internet. Since layering is one of the most common ways of dealing with complexity [1], the *layered-broker style* is proposed accordingly to tackle such challenge. In [19] the broker layer addressed the issue of heterogeneity when composing distributed web services. The authors argue that basic broker architectures cannot be directly applied to develop distributed web services since the web services mediation requires the homogenization of different service interfaces, a task needs layered structure with each layer focusing on each separate concern – remoting, mediating, and composing. The proposed layered-broker style can be depicted in Figure 3.

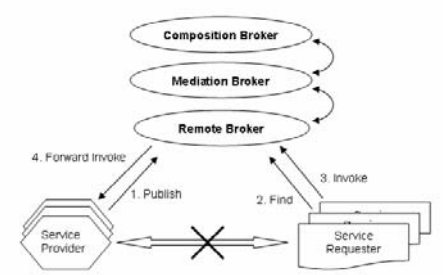


**Fig 3. Layered Broker Style**

### 2.4 Peer-to-Peer Style

Both matchmaker and broker styles reply on a central control point – the matchmaker or the broker. However, the peer-to-peer (P2P) architectural style assumes that it is not feasible to constantly rely on such a centralized, administratively managed infrastructure within an open environment (e.g. the Internet) where all the resources are connected and widely distributed [20]. Hence some research has introduced the peer-to-peer computing model into web services, thus structuring the peer-to-peer architecture style.

### 2.4.1 P2P Discovery Style (P2PD)

The most common P2P architecture in web services can be found in P2P based service discovery, which generally falls into two approaches. The first approach [21, 24, 25, 26] places the web services protocols above the native P2P protocols such as Gnutella [20], DHT [20], with WS-P2P adaptor to bridge the gap of two protocols.

The second approach [22, 23, & 27] constructs the P2P communication protocol using web services protocol. Meanwhile, both of these two approaches can also support the semantic-based services discovery [21, 24, 25, 27, 28]. These two approaches are illustrated in Figure 4.

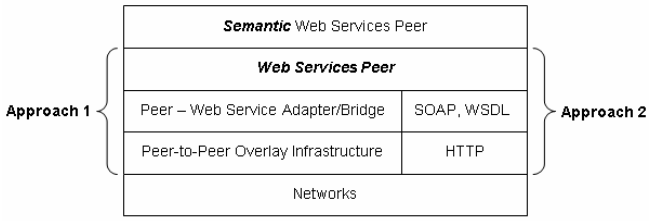


**Fig 4. P2P Discovery Structure**

### 2.4.2 P2P Composition Style (P2PC)

As indicated in Section 3.1 and 3.2, P2P execution (P2PE) is a common means to invoke web services in the matchmaker style. Hence in this section, we only focus on the P2P composition style, which can be further classified into three sub-styles.

**1) Static Composition Style (P2PC-S)**

In this style, overall process specification (e.g. BPEL4WS) is, at design-time, partitioned into small pieces and deployed to involved service providers. Such design-time partition approach is not new. The early study on partitioning process specification can be found in [32]. Project in [33] is the early work that uses such static partition to distribute web services processes. Authors in [34] present an algorithm to partition a single BPEL process into an equivalent set of decentralized processes. Based on [34], [29] proposed a decentralized BPEL composite scheme which contains multiple engines, with each executing composite web service specification at distributed locations. Figure 5 illustrates such static composition style.

**2) Mobil Composition Style (P2PC-M)**

Using this style, both process specification and instance with execution states are dynamically brought to the service providers at run-time. Authors in [35] utilize the mobile agent to encapsulate and deliver the process specification to each host where the desired services are invoked by such mobile agent. In architecture proposed by [31], the process engine which executes that service is also decided in an ad-hoc manner. The mobility of process is implemented using the message communication between peer engines. This style is depicted in Figure 6.

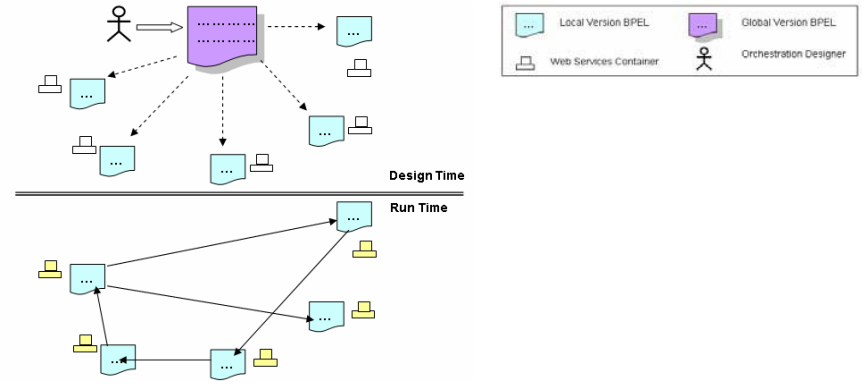


**Fig 5. P2P Static Composition**

**3) Hybrid Composition Style (P2PC-H)**

Project in [30] employed the *mobile P2P execution style* to create a true peer-to-peer service process execution runtime environment. The two-phase-commit protocol is used to distribute the process instance to the target node (service provider) based on the meta-information replicated from the global repositories. On the other hand, in order to achieve better performance and reduce the amount of data to be replicated, the authors also utilized the *static P2P execution style* to partition a process into a set of distributed execution units, which only contain small amount of data that can 1) execute the local service; 2) navigate the process according to the service invocation response.

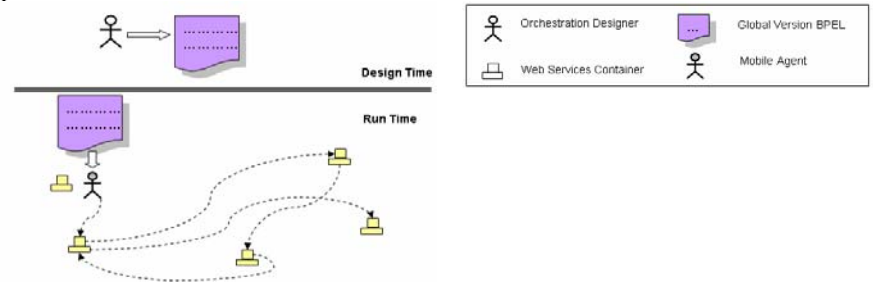


**Fig 6. P2P Mobile Composition**

## 3 Related Work

Research of architectural style and patterns has been extensively conducted in both academia and industry. Authors in [1] present eight architectural patterns that specify the fundamental structure of an application. However their result does not include web services application, which merely comes into view in recent years. A series of architectural styles for network application are surveyed in [8], which indicates that, given certain contexts, a specific architecture can be built by combing different archi-

tectural styles in a 'conflict-conciliated' manner. Again, this work focuses on web and network application, rather than web services application. Web services architectural patterns are defined and identified in [2]. While these patterns are limited in e-Business scenarios, they more or less reflect the business requirements of web services architecture from industry perspective. The formative research of architectural styles, which facilitates the design of SOA, still lies in its immature phase. Research in [5] is the closest work to our efforts. The authors catalog architectural styles that are essential for SOA applications and conceptually evaluate these identified six styles. Nevertheless, theses styles are based on their proposed multi-agent model rather than from related literature research and industry practices. Hence their work in this sense is different from the architectural style survey carried out in this paper.

## 4 Conclusions and Future Work

In this paper, we surveyed web services architectural styles in current literature. Our next research goal is to offer a comparison of these architecture styles based on a well-defined quality framework. The aim of such comparison is to help architect select the most appropriate architecture styles given specified requirements captured from the stakeholders.

## 5 References

1. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture, a System of Patterns. J. Wiley and Sons, Inc, 1996.
2. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T., 'Patterns: Service-Oriented Architecture and Web Services', IBM Redbooks, April 2004
3. Cimetiere, J.C. 'Web Services Adoption and Technology Choices – Analysis of survey results', Technical Report, Group SQLI, TechMetrix Research, 2003.
4. Lundberg, L., Bosch, J., Häggander, D. and Bengtsson, P. 'Quality Attributes in Software Architecture Design', Proceedings of the 3rd IASTED International Conference on Software Engineering and Applications, pp. 353-362, 1999.
5. Maximilien, E.M. and Singh, M.P., 'Toward Web Services Interaction Styles', In Proceedings of 2nd IEEE International Conferences on Services Computing, July 2005
6. K. Decker, K. Sycara, and M. Williamson. Middle agents for the internet. In Proceedings IJCAI-97
7. Jennings, Nick R., 'On Agent-Based Software Engineering', Artificial Intelligence, 117(2) pp. 277-296 (2000).
8. Fielding, R.T. 2000, 'Architectural Styles and the Design of Network-based Software Architectures', PhD Dissertations, University of California, Irvine CA, USA
9. Wong, H.C. and Sycara, K., 2000, 'A taxonomy of middle-agents for the Internet', Proceedings of the Fourth International Conference on MultiAgent Systems, July, 2000, pp. 465 - 466.
10. Wang, X., Yue, K., Huang, J. Z., Zhou, A., 2004, 'Service Selection in Dynamic Demand-Driven Web Services', Proceedings of the IEEE International Conference on Web Services (ICWS04)
11. Yu, T. & Lin, K. 2004, 'The Design of QoS Broker Algorithms for QoS-Capable Web Services, Proceedings of EEE'04
12. Degwekar, S., Su, S.Y.W., Lam, H. 2004, 'Constraint Specification and Processing in Web Services Publication and Discovery', Proceedings of the IEEE International Conference on Web Services (ICWS' 04)
13. Papazoglou, M. P., Kr¨amer, B. J., and Yang, J. 2003, 'Leveraging Web-Services and Peer-to-Peer Networks, Springer-Verlag Berlin Heidelberg 2003.

14. Pilioura, T., Kapos, G., Tsalgatidou, A. 2004, 'PYRAMID-S: A Scalable Infrastructure for Semantic Web Service Publication and Discovery', Proceedings of the 14th International Workshop on Research Issues on Data Engineering (RIDE'04)
15. Sivashanmugam, K., Verma, K., and Sheth, A., 'Discovery of Web Services in a Federated Registry Environment', Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE2004): Workshop on Ontology in Action, pp. 490-493, 2004
16. Paolucci, M., Soudry J., Srinivasan, N., and Semantic Sycara, K. 2004, 'A Broker for OWL-S Web Services', Proceedings of First Internatioinal Web Services Symposium, 22nd - 24th March, 2004
17. Malek, H.B. 2005, 'Service-Orientation: A Brief Introduction', OASIS SOA Reference Model, http://www.oasis-open.org/committees/download.php/12834/Service-Orientation.pdf, May 2005
18. Fuchs, M. 2004, 'Adapting Web Services in a Heterogeneous Environment', Proceedings of the IEEE International Conference on Web Services (ICWS'04)
19. Piers, P., Benevides, M., Mattoso, M. 2003, 'Mediating Heterogeneous Web Services', Proceedings of the 2003 Symposium on Applications and the Internet (SAINT' 03)
20. Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z. 2003, 'Peer-to-Peer Computing', Hewlett-Packard Company Technology Report
21. Ayyasamy, S., Patel, C., Lee, Y. 2003, 'Semantic Web Services and DHT-based Peer-to-Peer Networks: A New Symbiotic Relationship', Position Paper, School of Interdisciplinary Computing and Engineering University of Missouri – Kansas City
22. Wang, Q., Yuan, Y., Zhou, J., Zhou, A., 'Peer-Serv: A Framework of Web Services in Peer-to-Peer Environment', WAIM 2003, LNCS 2762, pp. 298 – 305, Springer-Verlag Berlin Heidelberg 2003
23. Prasad, V., Lee, Y. 2003, 'A Scalable Infrastructure for Peer-to-Peer Networks Using Web Service Registries and Intelligent Peer Locators', Proceedings of the 1st International Symposium on Cluster Computing and the Grid, p216, ISBN: 0-7695-1919-9
24. Paolucci, M., Sycara, K., Nishimura, T., Srinivasan, N. 2003, 'Using DAML-S for P2P Discovery', Proceedings of International Conference on Web Services, ISWS, 2003
25. Emekci, F., Sahin, O., Agrawal, D., and Abbadi, A. 2004, 'A Peer-to-Peer Framework for Web Service Discovery with Ranking', Proceedings of the IEEE International Conference on Web Services (ICWS'04), 0-7695-2167-3/04 IEEE
26. Schmidt, C. and Parashar, M. 2004, 'A Peer-to-Peer Approach to Web Service Discovery', World Wide Web, 7(2): 211-229, 2004
27. Banaei-Kashani, F., Chen, C-C., Shahabi, C., 2004, 'WSPDS: Web Services Peer-to-peer Discovery Service', Proceedings of International Symposium on Web Services and Applications (ISWS'04), Las Vegas, Nevada, USA, June 2004, pp 733-743
28. Thaden, U., Siberski, W., and Nejdl, W. 2003, 'A Semantic Web based Peer-to-Peer Service Registry Network', Technical Report, Learning Lab Lower Saxony, University of Hanover, Germany, 2003
29. Chafle, G., Chandra, S., and Mann, V. 2004, 'Decentralized Orchestration of Composite Web Services', In Proceedings of World Wide Web, May 17–22, 2004, New York, USA
30. Schuler, C., Weber, R., Schuldt, H., and Schek, H., 'Scalable Peer–to–Peer Process Management — The OSIRIS Approach', Proceedings of ICWS, 2004
31. Lakhal, N.B., Kobayashi, T., Yokota, H., 'THROWS: an architecture for highly available distributed execution of Web services compositions', Proceedings of the 14th International Workshop on Research Issues on Data Engineering:
32. Muth, P., Wodtke, D., Weissenfels, J., Kotz, D.A. 1998, 'From Centralized Workflow Specification to Distributed Workflow Execution', Journal of Intelligent Information Systems (JIIS), 10(2), 1998
33. Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A., 'Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services', Proceedings of ICDE, 2002
34. Nanda, M.G., Chandra, S., Sarkar, V. 2004, 'Decentralizing execution of composite web services', Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications, Volume 39, Issue 10, October 2004,
35. Haller, K. and Schuldt, H. 2003, 'Consistent Process Execution in Peer-to-Peer Information Systems', Proceedings of the 15th Conference on Advanced Information Systems Engineering (CAiSE), Klagenfurt/Velden, Austria, 2003

# Application of Data Engineering Techniques to Design of Message Structures for Web Services

George Feuerlicht

Faculty of Information Technology,
University of Technology, Sydney,
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
jiri@it.uts.edu.au

**Abstract**. Unlike components, Web Services are primarily intended for inter-enterprise applications that use Internet as the underlying transport mechanism, and consequently are subject to a different set of design considerations than intra-enterprise applications. Most practitioners recommend the use of coarse-grained, message-oriented Web Service that minimize the number of messages and avoid the need to maintain state information between invocations. In this paper we argue that excessive use of coarse-grained, document-centric message structures results in poor reuse and undesirable interdependencies between services. We describe a design approach that provides a framework for designing message payloads for service-oriented applications. We treat the problem of message design from a data engineering perspective and apply data normalization techniques to the design of service interfaces. We consider the impact of increasing message granularity on cohesion and coupling of service operations and discuss the associated design tradeoffs.

## 1 Introduction

Web Services are used mainly in inter-enterprise applications that rely on the Internet as the underlying transport mechanism, and consequently are subject to a different set of design considerations than intra-enterprise applications. Internet-scale distributed applications must be able to deal with unreliable connections, provider site failures, network latency, and trust issues. Most practitioners recommend the use of coarse-grained (i.e. services with aggregate data structures), message-oriented Web Services on the basis that coarse-grained Web Services generate fewer SOAP messages and therefore have lower communication overheads and less opportunity for failure [1]. Another motivation for using messages with aggregated message payloads is to avoid the need to maintain state information between individual requests. Proponents of the REST (Representational State Transfer) model [2] argue that Internet-scale distributed applications should rely entirely on stateless HTTP-based communications with coarse-grained XML message payloads. Given slow response times, high latencies, and poor reliability of the present Internet environment, performance is clearly an important consideration. However the design of Web Services should not be driven by performance-related objectives alone. The choice between fine-grained and

coarse-grained services is a choice between two extreme design options, each with different impact on performance as well as software engineering properties of service-oriented applications. Granularity (i.e. level of aggregation) of Web Services determines the scope of functionality that a given service (or service operation) implements, and is a key determinant of reusability and maintainability of service-oriented applications. While coarse-grained services achieve performance advantages by reducing the number of network interactions required to implement a given business function, when considered from the perspective of software engineering they suffer from a number of significant drawbacks, including limited reuse and poor maintainability. Coarse-grained Web Services are characterized by complex message structures that arise from designing message payloads to include all the information needed to perform the corresponding business function (e.g. airline booking). Complexity of messages is further increased by embedding business rules and constraints within the message data structures [3]. To illustrate this point consider, for example, the OTA (OpenTravel Alliance, www.opentravel.org/) flight booking business process. OTA defines request/response message pairs for individual business processes, and implements the flight booking business process using the OTA_AirBookRQ/OTA_AirBookRS message pair. The flight booking request document OTA_AirBookRQ (www.opentravel.org/downloads/2002B_XSD/OTA_AirBookRQ.xsd) is a complex, aggregate document that contains a large number of data elements (many optional) including flight booking, itinerary, traveler and payment details. The underlying assumption is that all of the information is available at the time of booking the flight, and that the airline tickets are paid for when the flight is booked. In practice, however flight booking and payment are often performed separately resulting in duplication of information and potential for data inconsistencies. The complexity and redundancy of message data structures makes it difficult to evolve the specification without producing undesirable side-effects that invalidate existing applications. Message payloads that externalize complex data structures result in high levels of data coupling and interdependencies between services, violating a fundamental design objective for distributed applications (i.e. minimization of coupling). Decomposing the flight booking request into separate, lower-granularity operations (e.g. flight enquiry, flight booking, payment, etc.) leads to simplification of the interface, improved flexibility, and potential for reuse (e.g. payment operation can be reused in another context, e.g. car rental, or a hotel room booking).

From a software engineering perspective, service interfaces need to be designed to maximize cohesion and minimize coupling [4], [5]. Maximization of cohesion refers to the requirement for methods to implement a single conceptual task and is closely related to reusability and maintainability of application components. High level of cohesion produces orthogonal services and improves the stability of the application as modifications can be typically confined to a specific service, or service operation. Minimization of coupling (i.e. interdependencies between services), results in improved ability to accommodate change. Applying these principles to service design leads to improved clarity of the interfaces, reduction in undesirable side effects, and improved flexibility of applications [6], [7]. Such requirements tend to favor finer-granularity services, and therefore conflict with performance considera-

tions. Balancing performance and software engineering considerations involves design tradeoffs and requires good understanding of the impact of service aggregation on cohesion and coupling of service-oriented applications.

In this paper we consider the problem of designing message structures for service-oriented applications from a data engineering perspective, applying data normalization rules to service interface parameters. In the following section (section 2) we briefly review research literature dealing with the design of Web Services applications. We then describe a framework for the design of message structures for service-oriented applications (section 3) and use this framework to design a set normalized interfaces (section 4). We then evaluate the impact of increasing message granularity on cohesion and coupling (section 5). In the concluding section (section 6) we summarize the benefits of the proposed design framework.

## 2 RELATED WORK

Web Services design approaches can be broadly classified into methodologies based on object-oriented design [8], [9], [10], [11], [12], methods for transformation of industry domain specifications [13], and business process transformation approaches [4], [14], [15], [16]. For example, Papazoglou and Yang [4] describe a design methodology that gives a set of service design guidelines based on the principles of minimizing coupling and maximizing cohesion to ensure that the resulting services are self-contained, modular, extendable and reusable. The methodology produces definition of WSDL Web Service interfaces and WSFL service flow models, and also includes non-functional service design guidelines that relate to service provisioning strategies and service policy management models. Web Services design is an active research area and while there is some agreement about the basic design principles there are no widely accepted design methodologies that can guide designers of Web Services applications. The focus of this paper is on the design of message data structures that form the basis of interaction between services and determine the software engineering properties of service-oriented applications.

## 3 DESIGN FRAMEWORK

From an architectural point of view service-oriented applications can be considered at different levels of abstraction. From one perspective they can be regarded as distributed systems that use message interchange as the basic communication mechanism, i.e. messages are regarded as the key artifacts of service-oriented applications. Message-oriented approaches and Message-Oriented Middleware (MOMs) have been used extensively in the context of Enterprise Application Integration (EAI) for the implementation of loosely-coupled, asynchronous applications. Alternatively, service-oriented applications can be viewed as programmatic environments that use procedure calls to execute local and remote procedures (RPCs). RPC-based programming environment is typically (but, not necessarily) used to implement synchronous,

tightly-coupled applications. We exploit this duality between messages and procedures and describe a design framework that leverages object-oriented design principles and data engineering techniques for the design of message structures for service-oriented applications.

### 3.1 Procedures vs. Messages

Procedures typically implement well-defined functions and use simple data parameters. However, it is possible to pass complex objects (e.g. XML documents) as procedure parameters, in effect using RPCs to interchange documents. Given this programmatic perspective, the interface contract is the signature of the corresponding procedure call (service operation), for example:

**FlightEnquiry**(INPUT: OriginLoc, DestinationLoc, DepartureDate,
OUTPUT: FlightNumber)

Given the message-oriented, document-centric perspective, message payloads (i.e. XML documents within SOAP envelopes) define the interface contract. For example, the XML schemas of the messages OTA_AirBookRQ and OTA_AirBookRS constitute the interface contract and specify the method signature as:

**BookFlight**(INPUT: OTA_Air_BookRQ,
OUTPUT: OTA_Air_BookRS)

Importantly, these abstractions are independent of the physical implementation of Web Services application that the designer may eventually choose. So that adopting the programmatic perspective during the design stage does not imply that the implementation of services will be based on synchronous RPCs. It is, for example, possible to conduct the design using the programmatic perspective and adopt the document style, asynchronous Web Service implementation. We regard decisions about the implementation style (i.e. binding style, RPC or document) and interaction model (i.e. synchronous or asynchronous, stateful or stateless) as orthogonal concerns to the task of designing the service interface, and defer such decisions to the implementation stage of the systems development process. This separation of concerns allows focus on interface design without introducing implementation dependent constraints during early design stages. From the design point of view, taking the document-centric perspective makes it difficult to reason about design tradeoffs associated with different message design strategies (e.g. level of message aggregation). However, changing the level of abstraction from messaging to programmatic interactions and regarding the messages structures as service interfaces makes it possible to apply well-established program design techniques to Web Services message payloads.

### 3.2 Design Principles

Using the programmatic perspective, the task of designing interfaces for service-oriented applications is conceptually similar to design of methods for object-oriented applications. The guiding principles for interface design include orthogonality (i.e. each interface should define a distinct function), maximization of method cohesion

and minimization of method coupling. Cohesion and coupling have been studied extensively in the context of structured and object-oriented programming [17], [18]. Myers [19] defined module cohesion as a degree of interaction within programming modules and coupling as the degree of interaction between programming modules, and classified both measures according to type. According to Myers, the highest levels of cohesion are Informational (all functions within a module share the same data) and Functional cohesion (module performs a single function). Minimal (i.e. the most desirable) types of coupling are Stamp coupling, where modules use data structures as parameters, and Data coupling where individual data elements are used as parameters. Thus the combination of Functional cohesion and Data coupling produces the most desirable situation from the point of view of reuse and maintainability. To achieve the highest level of cohesion the designer must ensure that service operations use the same data structures (i.e. Informational cohesion) and that each service operation (i.e. method) implements a well-defined, atomic task (Functional cohesion). Importantly, high level of method cohesion leads to orthogonality as functional overlap is minimized, or eliminated altogether. The requirement for data coupling dictates that interfaces consist of individual data parameters rather than complex data structures. Furthermore, using individual data parameters for interface specification rather than coupling via complex data structures (i.e. Stamp coupling) enables the application of data engineering techniques to minimize interdependencies between service operations, as described in the following sections.

### 3.3 Design Steps

The definition of service interfaces involves specification of operations and corresponding input and output parameters. This task is similar to designing method signatures in the context of object-oriented design, and involves identifying suitable candidate methods that are progressively refined to produce a set of well-defined service interfaces [5], [20]. The design framework consists of three design stages: initial design of service interfaces, refining interface design using interface normalization, and finalizing design by adjusting interface granularity. We base our design examples on the OTA the airline availability request/response messages: OTA_Air_AvailRQ/OTA_Air_AvailRS and booking request message pair: OTA_Air_BookRQ/ OTA_Air_BookRS.

Decomposition of the Flight Booking business function can be achieved by modeling the interaction between a travel agent and an airline using a Sequence Diagram. Each step in the Sequence Diagram dialog produces a Request/Response message pair and corresponds to an elementary business function [5]. Alternatively, elementary business functions can be identified as leaf functions in a business function hierarchy [21]. The resulting service interfaces correspond to elementary business functions as illustrated by the FlightEnquiry interface below:

**FlightEnquiry**(INPUT: OriginLocation, DestinationLocation, DepartureDate, OUTPUT: FlightNumber, DepartureAirport, ArrivalAirport, DepartureTime, ArrivalDate, ArrivalTime)

We can now apply interface normalization to detect extraneous interface parameters that can be removed in order to minimize data coupling between interfaces, and at the same time improve the cohesion of the operations.

## 4. Interface Normalization

Normalized data structures have been used extensively in database design [22]; we use the same principles here in order to minimize data coupling of service interfaces. Data coupling involves two or more interfaces being coupled via interface parameters, i.e. output parameters of one interface match input parameters of another. Removing data parameter interdependencies for the input and output parameter sets will ensure that both parameter sets are minimal (i.e. do not contain redundant parameters). We classify service operations according to type into query (i.e. operations that return data in output parameters given a query specified using input parameters) and update operations (i.e. operations that update data given update operation specified using input parameters), and formulate the following interface design rules [5]:

**Rule 1:** *Input parameters of query and update operations should form a minimal set, i.e. individual data parameters must be mutually independent*.
**Rule 2:** *Output parameters of query and update operations should form a minimal set, i.e. individual data parameters must be mutually independent.*
**Rule 3**: *Output parameters of query operations must be fully functionally dependent on input parameters.*

We regard the interfaces of query operation as relations where the input parameter set corresponds to the relation key, and the output parameter set are the non-key attributes. Output parameters of normalized interfaces are fully functionally dependent on the input parameter set, i.e. the interface parameters form a BCNF (Boyce-Codd Normal Form) relation. This ensures that parameters are used as data, not as control parameters and avoids Control coupling that involves using interface parameters to control the execution of the method [19]. Normalization of interfaces of query operations also ensures mutual independence of interfaces parameters for both input and output parameter sets (i.e. input and output parameter sets are minimal). Update operations, in general, do not exhibit functional dependencies between input and output parameters. However, both input and output parameters sets should be minimized by removing redundant data parameters, to avoid unnecessary data coupling. Now, assuming the functional dependencies below we can produce a set of to normalized interfaces:

FD1: {OriginLocation, DestinationLocation, DepartureDate → FlightNumber}
FD2: {FlightNumber → DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime}
FD3: {FlightNumber, DepartureDate → ArrivalDate}
FD4: {FlightNumber, DepartureDate, CabinType → Quantity}

FD5: {FlightNumber, DepartureDate, CabinType → BasicFareCode, BasicFare}

**Query Operations:**

**FlightEnquiry**(INPUT: OriginLocation, DestinationLocation, DepartureDate,
OUTPUT: FlightNumber )
**ScheduleEnquiry**(INPUT: FlightNumber,
OUTPUT: DepartureAirport DepartureTime, ArrivalAirport, ArrivalTime )
**ArrivalEnquiry**(INPUT: FlightNumber, DepartureDate,
OUTPUT: ArrivalDate)
**SeatEnquiry**(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: Quantity)
**PriceEnquiry**(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: FareBasisCode, BaseFare)

**Update Operations:**

**BookFlight**(INPUT: FlightNumber, DepartureDate, CabinType, TravelerName,
OUTPUT: BookingReferenceID)
**SeatingRequest**(INPUT: BookingReferenceID, SeatPreference,
OUTPUT: BookingReferenceID)
**MealRequest**(INPUT: BookingReferenceID, MealPreference,
OUTPUT: MealType)

We can verify that the interfaces are fully normalized by noting that all input parameters for the enquiry operations are determinants (i.e. right-hand side of functional dependencies) satisfying the condition for BCNF Normal Form [23]. In addition to minimizing coupling, the effect of interface normalization is to maximize cohesion as resulting interfaces implement atomic operations.

## 5. Finalizing Design

The above analysis leads to normalized service interfaces and results in fine-granularity operations. While this may be theoretically appealing, the associated increase in the number of runtime calls and complexity of the interaction dialogue makes this approach difficult to implement in practice given the existing low-reliability and slow response time Internet infrastructure. Finding an optimal level of granularity for Web Services and individual service operations requires further examination.

### 5.1 Adjusting Granularity of Interfaces

We can use the normalization framework introduced in section 4 to understand the impact of aggregating interfaces. For example, the query operations SeatEnquiry and

PriceEnquiry share common input parameters FlightNumber, DepartureDate, CabineType. Combining the two interfaces produces a composite operation SeatPriceEnquiry:

**SeatPriceEnquiry**(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: Quantity, FareBasisCode, BaseFare)

This clearly leads to loss of cohesion as the resulting operation no longer implements a single atomic task, and in situations where it is used to perform a partial enquiry (e.g. seat availability enquiry only) the operation returns values that are not used by the application. Applying the normalization framework, this lack of cohesion is reflected by a partial functional dependency between the input and output parameter sets of the SeatPriceEnquiry interface (i.e. the loss of *full* functional dependence). This tradeoff can be justified in this instance on the basis that both operations are frequently performed together, and that the benefits of reduced number of operations and runtime procedure calls outweighs the loss of cohesion. Similar considerations apply to update request operations. For example SeatingRequest and MealRequest can be combined into a composite operation SeatingMealRequest:

**SeatingMealRequest**(INPUT:BookingReferenceID, SeatPreference, MealPreference,
OUTPUT: SeatNumber, MealType)

This time, a partial request, e.g. seating request only, produces non-homogeneity with MealPreference and MealType left undefined.

### 5.2 Implementation Style and Interaction Model

Following decisions about the appropriate level of aggregation, the final design stage involves decisions about the implementation style (i.e. binding style, RPC or document) and interaction model (i.e. synchronous or asynchronous, stateful or stateless). Adopting the document-centric (message-oriented) approach the resulting interface definitions are transformed into document-style WSDL specifications. Alternatively, the resulting interfaces can be mapped directly into Web Services operations using the RPC binding style [3]. Detailed discussion of such implementation issues is outside the scope of this paper.

## 6. Conclusions

We presented a design methodology for Web Services that applies data engineering principles to the design of message structures of service-oriented applications. The design approach relies on the principles of orthogonality, maximizing method cohesion, and minimizing method coupling, and uses data normalization techniques to avoid externalization of redundant data parameters. While we have argued that excessive use of coarse-grained, document-centric message structures results in poor reuse

and undesirable interdependencies between services, we do not advocate fine granularity services as a universal solutions. Equally, this paper does not represent argument for any specific Web Services implementation style (i.e. RPC or document style), as such decisions need to be made in the context of specific application requirements and taking into account the implementation environment. The main benefits of the proposed design framework is that it facilitates making informed decisions about the level of granularity of service operations based on normalization of the underlying message structures. As shown in section 5, composite operations can be constructed from operations with fully normalized interface messages by combining operations based on the properties of interface parameters. The impact of the resulting message aggregation on cohesion and coupling of service-oriented applications can be evaluated using the normalization framework, so that the designer can determine the most appropriate message design for a particular set of requirements.

## 8. REFERENCES

[1] Huhns, Michael N. and Munindar P. Singh, "Service-Oriented Computing: Key Concepts and Principles," IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.

[2] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation, 2000, Available on: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] Feuerlicht, G. Implementing Service Interfaces for e-Business Applications. In Proceedings of the Second Workshop on e-Business (WeB 2003), Seattle, USA, December 2003. ISSN: 1617-9846.

[4] Papazoglou, M.P. and Yang, J. (2002), Design methodology for Web services and business processes. In Proceedings of the 3rd VLDB-TES workshop (Hong Kong, August, 2002). Springer, pages 54-64.

[5] Feuerlicht, G, Designing Service-Oriented e-Business Applications using Data Engineering Techniques, The Third Workshop on e-Business, in conjunction with ICIS 2004, December 11, 2004, Washington D.C., USA, ISBN:957-01-9161-9

[6] Venners, B. (1998) Introduction to Design Techniques. Available on: http://www.javaworld.com/javaworld/jw-02-1998/jw-02-techniques.html, February, 1998.

[7] Venners, B. (2002) API Design: The Object. Available on: http://www.artima.com/apidesign/object.html, April 26, 2002.

[8] Ambler, S.W. (2002) Deriving Web Services from UML models, Part 1: Establishing the process. Available on: http://www-106.ibm.com/developerworks/webservices/library/ws-uml1/

[9] Levi, K. and A. Arsanjani (2002) A goal-driven approach to enterprise component identification and specification. Communications of the ACM. Vol. 45:(10). (2002) 45 - 52

[10] Luo, M. et al. 2005, Service-Oriented Business Transformation in the Retail Industry Part 1: Apply SOA to Integrate Package Solutions and Legacy Systems [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-retail1/ [Accessed 15 April 2005].

[11] Meyer, B. Object-oriented Software Construction. 2nd ed. Prentice Hall, Upper Saddle River, N.J., 1997.

[12] Smith, R. Modeling in the Service Oriented Architecture, 2003. http://archive.devx.com/javasr/articles/smith1/smith1-1.asp.

[13] Masud, S. RosettaNet-based Web Services, Part 2: BPEL4WS and RosettaNet, 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-rose2/.
[14] Leymann, F. Web Services Flow Language (WSFL 1.0), 2001.http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.p
[15] Radeka, K. Designing a Web Services Project for Maximum Value: the 90 Day Challenge. In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications archive (OOPSLA 2002) Practitioners Reports, Seattle, Washington, November 2002. ACM Press New York, NY, USA. ISBN:1-58113-471-1.
[16] Stevens, M. Multi-Grained Services. http://www.developer.com/design/article.php/1142661, May 21, 2002.
[17] Yourdon, E. and Constantine, L.. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice-Hall, Englewood Cliffs, N.J., 1979.
[18] Stevens, W.P., Myers, G.J., and Constantine, L.L., Structured Design, IBM SYSTEMS JOURNAL, VOL38, NOS2&3, 1999
[19] Myers, G.J.: Composite Structured Design, 1978, Van Nostrand Reinhold, ISBN 0-442-80584-5, 175 pages
[20] Feuerlicht, G., Design of Service Interfaces for e-Business Applications using Data Normalization Techniques, Journal of Information Systems and e-Business Management, Springer-Verlag GmbH, 26 July 2005, pages 1-14, ISS:1617-98
[21] Feuerlicht, G. and S. Meesathit. Design Framework for Interoperable Service Interfaces. In Proceedings of the 2nd International Conference on Service Oriented Computing, pp. 299-307, New York, NY, USA, November 2004. ACM Press. ISBN:1-58113-871-7.
[22] Codd, E.F (1971). Normalized Data Structure: A Brief Tutorial. In Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control (San Diego, California, November 11-12, 1971). ACM, 1971, 1-17.
[23] Date, C. J. Fagin, R. (1992) Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases, ACM Transactions on Database Systems (TODS) Volume 17, Issue 3 (September 1992) Pages: 465 - 476, 1992, SSN:0362-5915

# A Consensus-Based Service Discovery

Chun-Lung Huang[1], Ping Wang[2], Kuo-Ming Chao[3], Chi-Chun Lo[1], Jen-Yao Chung[4]

[1] Institute of Information Management, National Chiao Tung University, Taiwan
{clhuang@iim,cclo@faculty}.nctu.edu.tw
[2] Department of MIS, Kun Shan University of Technology, Taiwan
pingwang@mail.ksut.edu.tw
[3] DSM Research Group, School of MIS, Coventry University, Coventry, CV1 5FB, UK
k.chao@coventry.ac.uk
[4] IBM T. J. Watson Res. Center, P.O. Box 218, R. 05-15, Yorktown Heights, NY, 10598, USA
jychung@us.ibm.com

**Abstract.** Efficient service discovery is a crucial process for web service composition. Most existing service discovery approaches, based on either functional or non-functional attributes, do not address the issues associated with the impact of subjective expectations and preferences on service discovery. For instance, searching for a Web service (for cheap and convenient flights), does not only involve functional requirements, but it also includes subjective and fuzzy opinions on query terms such as "cheap and convenient flights". However, different perceptions of such terms by service consumers and providers create serious problems in web service discovery. This is compounded by the fact that they may have different preferences or opinions. This paper presents a moderated fuzzy web service discovery approach to model subjective and fuzzy opinions, and to assist them in reaching a consensus. The method achieves a common consensus on the distinct opinions and expectations of individual users. This process is iterative and further fuzzy opinions and preferences can be added to improve precision in web service discovery.

## 1. Introduction

The emergence of web services has created unprecedented opportunities for organizations to reshape the landscape of collaboration with other organizations, using more flexible and dynamic means. The realization of value-added services by composing existing ones is gaining significant momentum. The success of web service composition significantly relies on mechanism for service discovery. Many available web services provide overlapping or identical services in terms of functionality, but with different Contents of Services. It is essential to make choices to determine which services are to participate in a given composite service. As an example, a travel agent is required to find a cheap and convenient flight to destination A using various airlines (web services). These airline web services provide the same or similar functions, but with different degrees of quality of services and various contents in their services. The introduction of a discovery mechanism to the process could increase the effectiveness. The premise is that the contents of data repositories for web services should be sum-

marized and represented at higher levels of abstraction (that is the information quality or quality of services for each web service). By looking up the summarized information, the discovery mechanism could filter out those services which are less conformed to consumers' anticipation and locate the desired services with higher quality of services. This approach can increase the likelihood of discovering acceptable service efficiently, if the precision (probability) is acceptable and the number of recommended services from the discovery mechanism is reasonable.

The improvement can be achieved by introducing an effective advertisement mechanism for the contents of services with a high level of abstraction or fuzzy terms. The precision lies in the representation of the fuzzy terms, which need to be defined at least loosely and consistently among service providers and consumers. However, the difficulty of having consistent terms among both sides is notorious, since they have different expectations and experiences. This is complicated by the differences among their preferences over the criteria. This research attempts to alleviate these problems. This paper mainly focuses on the report of a consensus-based web service discovery architecture that enables consumers and providers to reach a consensus on their expectations and preference criteria over the terms they use, if there are any. The resulting system should increase the success rate for a web service consumer in discovering the required services. Therefore, the service providers can improve the quality of services.

## 2. A Moderated Fuzzy Web Service Discovery Architecture

In order to achieve an effective web service discovery, a number of components, including fuzzy classifier, fuzzy engine, UDDI, OWL-S, fuzzy discovery and fuzzy moderator, are incorporated in the proposed moderated fuzzy web service discovery architecture (or called Moderated Fuzzy Discovery Method / MFDM) (see Fig. 1).

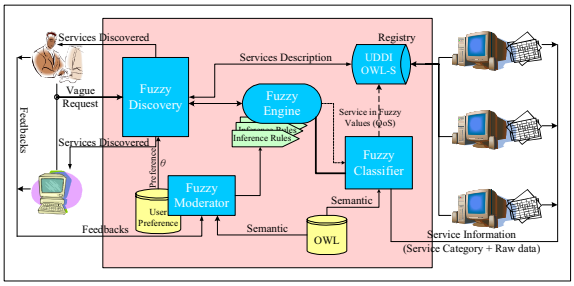


**Fig. 1. The Proposed Architecture for Moderated Fuzzy Web Service Discovery**

The fuzzy classifier is used to summarize and represent the content of each web services at higher levels of abstraction after browsing the data repositories. It contains essential predefined knowledge for interpreting and classifying the information residing in web services and consists of primitive and composite fuzzy terms, modifier and quantification fuzzy terms, and fuzzy rules (inference rules). Primitive terms are a set of atomic terms that represent a collection of raw data. Composite terms are generated through the combination of compound statements (primitive terms) and fuzzy rules.

The composite terms can also be represented in fuzzy rules, when heuristic associations between terms are required. The quantification terms are used to model the probabilities of occurrences. A statement can be signified by a modifier to make the statement a little vague. In other words, the statements associated with quantification and modifier terms are represented in fuzzy rules for the purpose of reasoning. The fuzzy classifier extends the aforementioned rules and their combinations to provide powerful classifications for each web services and gives each of them a value of information quality (QoS) as the higher level of abstraction. The terms, which are represented in Effect of Process class of the OWL-S [9], are declarative facts supported by other fuzzy terms and rules. The supporting fuzzy rules and sets are considered as ontologies represented in OWL for further reasoning.

A fuzzy engine is used to drive the fuzzy classifier and to carry out the classification process in order to evaluate the values of quality of service for each web service in the same specific service domain. After the classification process, at least one QoS value will be appended to the advertisement information registered in the UDDI and OWL-S database for each web service. This QoS value provides the fuzzy discovery with information to screen out the insignificant web services and help the consumers to locate the desired services when vague query requests occur.

The proposed framework, Moderated Fuzzy Web Service Discovery Architecture, adopts standard UDDI as a tool for advertising web services. However, the information represented in UDDI lacks well-defined meaning, so it cannot fully support computers and people to work in cooperation. With the complimentary support from semantic web technology, the descriptions in UDDI can be modeled in OWL-S and OWL. Retaining a list of semantic webs in UDDI provides a convenient way to discover web services, as the grounding profile in OWL-S is able to locate WSDL documents and the associated web services.

A fuzzy discovery also provides a function that converts crisp requests from service consumers into fuzzy requests. It is important to have the crisp terms transformed into fuzzy terms for the use of approximate reasoning, as contents of services have been represented in fuzzy terms. The fuzzy discovery also allows the consumers to use linguistic qualifier such as *more*, *less*, etc for searching web services. The detailed descriptions on the fuzzy discovery method can be found in our previous work [6].

The fuzzy moderator is a mechanism to assist the service consumers and providers in reaching consensus on the terms they use and the preferences over different criteria. It is assumed that the web consumers and providers possess different opinions and preference on the services they are about to consume or provide. The opinions and preference, on which most or majority of users can agree in the group, will be acceptable to the other members. The fuzzy moderator is able to incorporate iteratively users' subjective opinions and preferences and transform them to less subjective ones. In principle, the more feedbacks from users, the less subjective. The detail components and steps for fuzzy moderator will be elaborated in the following section.

A number of tools are used for the implementation. The web services are implemented via JAXRPC and their associated database systems are designed in MS Access. The ontologies are defined through Protégé. OWLJESSKB, which is able to interpret OWL syntax, is employed for reasoning. Extra functionalities are added to OWLJESSKB in order to reason about fuzzy rules and sets.

## 3. Fuzzy Moderator

The precision rate for service discovery is critically influenced by the fuzzy rules (inference rules) in the fuzzy classifier which are used to summarize the content of web services (i.e. cheap or convenient). The more the inference rule conforms to the consumer's expectation, the higher the precision rate is. Initially arbitrary opinions and preferences are adopted for the construction of default inference rules. However, these default rules might not conform to consumer's expectation and it could lead to the unsatisfactory precision rate.

The goal for fuzzy moderator is to find the group consensus on service terms (criteria) and allows derived consensual values to replace the default ones for acquiring the better classification results and precision rates in discovery. A fuzzy moderator includes two main sub-components: (1) the Similarity Aggregation Method (SAM) [5] and (2) the Resolution Method for Group Decision Problems (RMGDP) [2],[3],[4]. During the moderation process, the SAM will be triggered first to gain the group consensus on each of the criterion and subsequently RMGDP will be invoked to obtain the group preferences on different criteria.

### 3.1. Similarity Aggregation Method (SAM)

The adoption of the Similarity Aggregation Method (SAM) is to resolve different opinions among service consumers and providers. SAM is the method that can aggregate different users' fuzzy opinions to reach a group fuzzy consensus opinion. The method employs the similarity measure to calculate the difference between one individual with the others within the group in order to obtain the index of consensus. The index of consensus for each individual can be collected as a set and calculated to form an agreement among the group. After the process of SAM, the new consensual cognition about specific fuzzy terms will be derived to replace the default values in the fuzzy classifier. With the use of SAM, the cognition of fuzzy terms between consumer group and service providers can be relatively consistent, so the new QoS value can be more representative. The detailed procedures can be found in [5] and the efficiency was proved by the experiment in the previous work [1].

### 3.2 Resolution Method for Group Decision Problems (RMGDP)

The objective of RMGDP [2],[3],[4] is to reach group consensus on preference over different criteria based on major opinions in the group. It can be divided into three steps as following: (a) transformation process, i.e., to transform the individuals' opinions into preference values (the uniform representation), (b) aggregation process, i.e., to aggregate the individual preference values over different criteria to obtain the group preference for all consumers in the group using OWA (Ordered Weighted Averaging) operator [11], and (c) exploitation process, i.e., to compute the ranking of the alternatives by group preference. The resolution method for group decision problems [2],[3],[4] is summarized as follows:

***(a) The transformation phase:***

First, a collection of users have to be formed as a group. Each user has to evaluate alternatives according to the defined criteria, and then assign ordering preference to the alternatives for each criterion individually. The users allocate orderings based on their own preferences and subjective judgments. A transfer function is applied to convert those individual ordering of alternatives to a uniform representation, which characterizes the ordering preference degree between alternative $a_i$ and $a_j$ of $User_k$.

***(b) The aggregation phase:***

The collective preference is an aggregation of the users' ordering preferences obtained by the means of fuzzy majority [4]. The fuzzy majority is the product of combining the OWA operator with the fuzzy quantifier. The merging function of the OWA operator and the fuzzy quantifier $Q$ infers the collective ordering preference on each alternative. This process helps to aggregate all opinions into one consensus result.

***(c) The exploitation phase:***

The exploitation process is a consequence of identifying the priority of alternatives of group preference. In this process, two well-known and complimentary fuzzy ranking methods are used: (1) Quantifier Guided Non-Dominance Degree (*QGNDD*) and (2) Quantifier Guided Dominance Degree (*QGDD*) [13]. These two indexes are able to prioritize the final collective ordering preference and result the weightings under group consensus.

## 4. A Numerical Example

This section mainly illustrates the resolution process for fuzzy moderator in the context of the proposed architecture. The QoS term: *satisfaction* is the inference rule used by the fuzzy classifier for summarizing the information of each web service at a higher level of abstraction and is used for the vague query request about finding a Web service (for cheap and convenient flights) by the consumers. QoS term: *satisfaction* denoted as satisfaction($\widetilde{Q}$) is a composite term results from the following primitive inference rules: (1) QoS term: *cheap* is a measurement of the cost that is defined as cheap($\widetilde{Q}$), and $\widetilde{C}$ is its shorthand. (2) QoS term: *seatsize* is a scale for the available space of the seat represented as seatsize($\widetilde{Q}$) or $\widetilde{S}$ for short. (3) QoS term: *airtime* represents the length of flight time denoted as airtime($\widetilde{Q}$) or $\widetilde{T}$ for short.

So, the degree of *satisfaction* can be obtained by assigning them with default equal weightings and adding them up, i.e. $\widetilde{Q}_{init}$ = 1/3 × $\widetilde{C}_{init}$ + 1/3 × $\widetilde{S}_{init}$ + 1/3 × $\widetilde{T}_{init}$. $\widetilde{Q}_{init}$ is the default inference rule used for classification. After classification, consumers can use vague query supplied by MFDM for quick find the flights with satisfaction (cheap and convenient). This default inference rule, however, may not conform to consumers' expectation so the precision rate is not expected. For better precision rate, the initial value for $\widetilde{C}, \widetilde{S}, \widetilde{T}$ and weightings need to be modified to reflect the situation after a number of users' feedbacks have been collected and calculated by the proposed method.

We assume that there is a group of consumers, denoted as $User_i (i = 1,2,3,..., m)$, with their different subjective opinions on the definition of the term *cheap*. When they use the fuzzy queries, their feedbacks on the terms *cheap, seatsize* and *airtime* can be denoted as $\widetilde{C}_i(a_i, b_i, c_i, d_i)$, $\widetilde{S}_i(a_i, b_i, c_i, d_i)$ and $\widetilde{T}_i(a_i, b_i, c_i, d_i)$ as following:

$\widetilde{C}_1(a_1, b_1, c_1, d_1) = (0,0,13500,16500)$, $\widetilde{S}_1(a_1, b_1, c_1, d_1) = (1.5,2,2.5,2.5)$, $\widetilde{T}_1(a_1, b_1, c_1, d_1) = (0,0,2.5,2.5)$,
$\widetilde{C}_2(a_2, b_2, c_2, d_2) = (0,0,14500,14500)$, $\widetilde{S}_2(a_2, b_2, c_2, d_2) = (1,1,2,2)$, $\widetilde{T}_2(a_2, b_2, c_2, d_2) = (0,0,2.5,3.5)$,
$\widetilde{C}_3(a_3, b_3, c_3, d_3) = (0,0,14000,15500)$, $\widetilde{S}_3(a_3, b_3, c_3, d_3) = (0.8,1,2,3)$, $\widetilde{T}_3(a_3, b_3, c_3, d_3) = (0,0,1.8,2.8)$,
$\widetilde{C}_4(a_4, b_4, c_4, d_4) = (0,0,11000,13000)$, $\widetilde{S}_4(a_4, b_4, c_4, d_4) = (0.6,1.2,1.8,2.5)$, $\widetilde{T}_4(a_4, b_4, c_4, d_4) = (0,0,1.5,3)$

After SAM processed these feedbacks, a moderated fuzzy set for QoS term: *cheap*, $\widetilde{C}(0,0,13314.333,14925.007)$, is obtained to replace the existing one ($\widetilde{C}_{init}$). Applying the same principle, we can gain the $\widetilde{S} = (0.8911,1.2008,2.0105,2.5181)$ and $\widetilde{T} = (0,0,2.0666,2.9379)$ for QoS term: *seatsize* and QoS term: *airtime* to replace the existing $\widetilde{S}_{init}$ and $\widetilde{T}_{init}$ respectively. The fuzzy engine can use the less subjective value ($\widetilde{C}, \widetilde{S}, \widetilde{T}$) to evolve in order to improve the quality of service discovery. The aggregated values for each criterion are moderated but the weightings to the criteria are not. RMGDP is utilized to assist users in reaching consensual weightings of $\widetilde{C}$, $\widetilde{S}$ and $\widetilde{T}$.

Table 1: QGDD, QGNDD and Consensus Weightings for Alternatives

| QGDD for alternatives | $a_1$ (cheap) | $a_2$ (seatsize) | $a_3$ (airtime) | consensus weights for alternatives from QGDD | $wa_1$ (cheap) | $wa_2$ (seatsize) | $wa_3$ (airtime) |
|---|---|---|---|---|---|---|---|
| | 0.5938 | 0.4063 | 0.5000 | | 0.3959 | 0.2708 | 0.3333 |
| QGNDD for alternatives | $a_1$ (cheap) | $a_2$ (seatsize) | $a_3$ (airtime) | consensus weights for alternatives from QGNDD | $wa_1$ (cheap) | $wa_2$ (seatsize) | $wa_3$ (airtime) |
| | 1 | 0.8125 | 0.9375 | | 0.3636 | 0.2955 | 0.3409 |

Assume that each consumer provides his / her preferences on alternatives *A* using a preference ordering $O_s^k = \{o_1^k, o_2^k, ..., o_n^k\}$ (*n* is the number of alternatives). For example, consider that four consumers, $User_k (k = 1,2,3,4)$, provide their preferences on alternatives $A = \{a_1, a_2, a_3\}$, where $a_1$ is *cheap*, $a_2$ is *seatsize*, and $a_3$ is *airtime*, by the following ordering $O^1 = \{a_3, a_1, a_2\}$, $O^2 = \{a_1, a_3, a_2\}$, $O^3 = \{a_1, a_2, a_3\}$ and $O^4 = \{a_2, a_3, a_1\}$. After RMGDP, the consensus of four consumers is reached. The order of importance of three alternatives can be observed evidently from the part on left of Table 1 which generated from QGDD or QGNDD. We can then conclude that the importance of 3 criteria is $a_1$ (cheap) > $a_3$ (airtime) > $a_2$ (seatsize). The value of QGDD and QGNDD can be used to calculate the weightings for each alternative. If $W = \{wa_1, ..., wa_n\}$ is a weighting vector and $wa_i$ denotes the consensus weightings for alternative *i*, $wa_i = a_i / \sum_{i=1}^{n} a_i$, $wa_i \in [0,1]$, and $\sum_{i=1}^{n} wa_i = 1$. The consensus weightings for alternatives derived from QGDD and QGNDD are formulated as $W = \{0.3959, 0.2708, 0.3333\}$ and $W = \{0.3636, 0.2955, 0.3409\}$, that is the consensus weighting for QoS term: *cheap* and its value is 0.3959 (the upper right part of Table 1). Finally, the QoS term: *satisfaction* can be moderated as $\widetilde{Q} = 0.3959 \times \widetilde{C} + 0.2708 \times \widetilde{S} + 0.3333 \times \widetilde{T}$ to replace the default inference rule used for classification of web services.

## 5. Experimental Results

A case study with four different service consumers and ten airline service providers was adopted to evaluate three different methods namely Capability Discovery Method (CDM), Fuzzy Discovery Method (FDM) [6] which uses the default inference rule, and Moderated Fuzzy Discovery Method (MFDM) which uses the moderated inference rule. In order to examine their overall performances, three different sets of experiments were carried out and each set contain 10 experiments in order to gain their average precision rates. 34 fuzzy terms such as very cheap, most available, comfortable etc and their associated rules between web service providers and consumers were designed to represent their requests and underlying data repositories for the use of fuzzy and moderated fuzzy discovery methods.

After 30 experiments have been carried out and the result of each experiment was recorded, each user's satisfactory rates to the recommended services were classified and averaged according to three different methods for the investigation of their precision rates. Fig.2 shows that the MFDM with 75% precision rate has the best performance. FDM has produced correct recommendations just about over than half. CDM only has 40% precision rate. We can conclude that the proposed MFDM has outperformed the FDM and the FDM has produced better precision rate than the CDM. The MFDM has performed nearly twice better than the CDM in terms of precision rate.

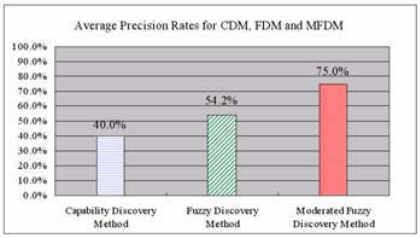


**Fig. 2.** Average Precision Rates for CDM, FDM and MFDM

## 6. Discussion and Summary

From the experimental result, the proposed Moderated Fuzzy Discovery Method (MFDM) has demonstrated that it is an effective method in web service discovery. However, there are a number of lessons we have learned from these. It is a non-trivial task to collect and classify the information and represent them appropriately in fuzzy terms. So, collecting the information from various airline websites and building them into a database were conducted. It is the same problem with collecting users' preference and opinions. This leads to that the number of web services were created is relative small. This affects the output of preference weightings which did not produce the great differences from the original one. The scalability issue will be tackled in the future. Since the fuzzy majority method was adopted for reaching consensus, we assume that users will change their opinions and preferences in line with the consensus. This is may not be the case when users have strong opinions and preferences. A nego-

tiation system will be in place to resolve this issue. We believed that this method is complimentary to [7],[8],[12] as it introduces another dimension to the web service discovery based on QoS. Research [10],[14] on the web-based database have made great progress on the query techniques and categorizing correlation among databases, but the consensus issue has not been addressed. The proposed method could provide a valuable mechanism to increase their precision rate.

The main contribution of this work is that it presents a moderated fuzzy web service discovery mechanism which allows web service providers and consumers to reach consensus on contents of services, even though they have different opinions and preferences. As a result, the proposed method can improve precision in service discovery. A number of experiments have been carried out to demonstrate that the proposed method outperforms capability based and traditional fuzzy discovery methods.

## References

1. C-L Huang, K-M Chao, C-C Lo: A Moderated Fuzzy Matchmaking for Web Services, (to appear in) Proceedings of The 5th International Conference on Computer and Information Technology, Shanghai, China, IEEE CS, (2005)
2. E Herrera-Viedma, F. Herrera, and F. Chiclana: A Consensus Model for Multiperson Decision Making With Different Preference Structures. IEEE Transactions on Systems, Man and Cybernetics, IEEE, Vol. 32. (2002) 394–402
3. F. Chiclana, F Herrera, and E. Herrera-Viedma: Integrating Multiplicative Preference Relations in A Multipurpose Decision-making Model Based on Fuzzy Preference Relations. Fuzzy Sets and Systems, Elsevier Science, Vol. 122. (2001) 277–291
4. F. Chiclana, F. Herrera, E. Herrera-Viedma: A Classification Method of Alternatives for Multiple Preference Ordering Criteria Based on Fuzzy Majority. Journal. Fuzzy Math., Vol. 34, (1996) 224–229
5. H-M Hsu, C-T Chen: Aggregation of Fuzzy Opinions under Group Decision Making. Fuzzy Sets and Systems, Elsevier Science, Vol. 79. (1996) 279-285
6. K-M Chao, M. Younas, C-C Lo, T-H Tan: Fuzzy Matchmaking for Web Services. Proceedings of 19 IEEE Conf. on Advanced Network and Inform. Appl., IEEE CS, (2005) 721-726
7. L. Zeng, B. Benatallah, A.H.H Ngu, M. Dumas, J. Kalagnanam, H Chang: Qos-Aware Middleware for Web Service Composition. IEEE Transactions on Software Engineering, Vol. 30, Issue 5. IEEE, (2004) 311–327
8. M. Lin, J. Xie, H. Guo: Solving QoS-Driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction. Proceedings of 2005 IEEE International Conf. on e-Technology, e-Commerce and e-Service (‘EEE 05), IEEE CS Press, Hong Kong (2005) 9–14
9. OWL Services Coalition “OWL-S: Semantic Markup for Web Services” OWL-S v. 1.1, White Paper, http://www.daml.org/services/owl-s/1.1/, Nov. (2004)
10. R Fagin, R Kumar, D. Sivakumar: Efficient Similarity Search and Classification via Rank Aggregation. Data Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM Press, (2003) 301 – 312
11. R.R. Yager: On Ordered Weighted Averaging Aggregation Operators in Multi Criteria Decision Making. IEEE Trans. on Sys., Man and Cyber., IEEE, Vol.18. (1988) 183-190
12. S Ran: A model for Web Services Discovery with QoS. ACM SIGecom Exchange, Vol. 4, Issue 1. (2003) 1-10
13. S.A. Orlovski: Decision-Making with A Fuzzy Preference Relation. Fuzzy Sets and Systems, Elsevier Science, Vol.1. (1978) 155-167
14. U Nambiar, S Kambhampati: Answering Imprecise Database Queries: A Novel Approach. Proceedings of the 5th ACM International Workshop on Web Information and Data Management, ACM Press, (2003) 126 – 133

# Specifying Reference Styles for Service Orchestration and Composition *

Karim Guennoun and Khalil Drira

LAAS-CNRS, 7 Avenue du Colonel Roche 31077 Toulouse Cedex 4 FRANCE,
{guennoun,khalil}@laas.fr

**Abstract.** In this paper we introduce basic architectural styles for the design of service-oriented applications. For this purpose we develop an appropriate formal framework using graph grammars. Our approach enables either generating architectures in conformance with a given style or checking conformance of ad-hoc architectures. Here we describe formally the basic interaction style involving elementary interactions between a service requestor and a service provider. Then we consider the orchestrated interaction style where an orchestrator manages the workflow of several service requestors and providers. Finally, we define a complex architectural style to address the compositional aspect of service-oriented architectures introducing the notion of composite services. We then compose the previous styles to define the composite basic invocation style and the composite orchestrated style.

## 1 Introduction

Formalization constitutes an important issue for the correct design of service-oriented applications. Existent research and standardisation on service-oriented applications and web services focus on behavioural aspects (workflow) and service description (interface specification). We propose, here, to extend the formal scope by studying the architectural characteristics within a formal framework based on graph grammars [1]. We believe that this will be helpful, as for component-oriented architectures and the related architecture description languages (ADLs), to facilitate the description, comprehension, and the verification of service-oriented architectures.

We present in this paper an approach for the architectural description of service-oriented applications. We illustrate our approach by specifying the basic architecture styles. We introduce the basic interaction style involving elementary interactions between a service requestor and a service provider. Then we specify the orchestrated interaction. Within this architectural style, we introduce the orchestrator component which is dedicated to manage the workflow between a set of service requestors and providers. After, we consider service composition and introduce the composite style considering composite service providers. Finally,

* This work has been developed within the framework of the IST project WS-DIAMOND

based on the combination of the two first styles and the composite one, we specify formally the composite basic interaction and the composite orchestrated interaction styles concerned with both coordination and composition.

## 2 Describing service-oriented architecture styles using the graph formalism

Graphs are a comprehensive and intuitive formalism to describe the architecture of service-based systems. Indeed, it is common to consider that vertices represent services and edges correspond to their related interdependencies. The use of this formalism becomes even more relevant when we address the specification of architectural styles since the declarative aspects (corresponding to the description of all the possible instances) can be correctly specified by graph grammars. This allows having a general formalism independent of any implementation language and a formal base to specify and check the constraints on these architectures. Several work was undertaken within this framework especially in the field of component-oriented architecture description among which we can quote [2] for the coordination of dynamic architectures, [3] for the description of architectures and their communication and [4] for description, diagnosis and repair.

We introduce here our model of grammar productions by extending a traditional approach called *single pushout* [5]. The grammar productions belonging to this basic model are of form $< L, R >$ where $L$ is a sub-graph to be identified and removed in the host graph (graph to which the rule will be applied) while $R$ constitutes the part to be added.

The extension introduced here consists in specifying a sub-graph $K$ of the pattern $L$ that will be maintained after the production application. Consequently, the grammar productions will be described by a triplet $\langle L; K; R \rangle$. A production of this type is applicable to a graph $G$ if there is an occurrence of $L$ in $G$. If so, It is applied by removing the pattern $L \backslash K$ from $G$ and by adding a copy of pattern $R$. Suppression of $L \backslash K$ consists in removing all the vertices and edges belonging to $L \backslash K$, and all the edges which bind a vertex in $L \backslash K$ with a vertex in $(G \backslash (L \backslash K))$ . A graph grammar is thus a system $\langle AX; NT; T; P \rangle$, where $AX$ is the axiom, $NT$ is the set of the non-terminal vertices, $T$ the set of terminal vertices, and $P$ the set of grammar productions. An instance belonging to the graph grammar is a graph containing only terminal vertices and obtained starting from axiom AX by applying successively productions in P.

## 3 Basic Interaction (BI) architecture style

The BI architecture style can be considered as the basic one for service-oriented architectures. Figure 1 presents the minimalist infrastructure composed of a service requestor and a service provider exchanging, for example, SOAP messages on the top of Internet application layer protocols.

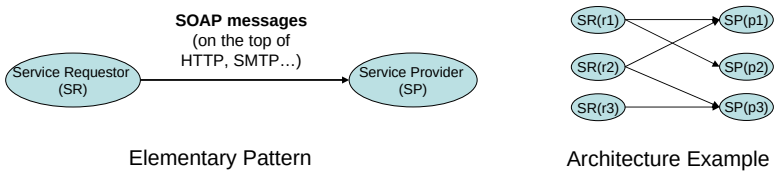


**Fig. 1.** Elementary pattern and architecture example of the direct interaction service-oriented style

This style of architecture is described by the graph grammar:
$\langle AX; \{BI\}; \{SR, SP\}; P_{BI} \rangle$ Where $P_{BI}$ is the following set of productions:

$\langle$ { AX } ; { } ; { BI } $\rangle$ $(BI.1)$
$\langle$ { BI } ; { } ; { } $\rangle$ $(BI.2)$
$\langle$ { BI } ; { BI } ; { SR(r) , SP(p), (r$\rightarrow p$)}$\rangle$ $(BI.3)$
$\langle$ { BI , SR(r) } ; { BI , SR(r) } ; { SP(p) , (r$\rightarrow p$)}$\rangle$ $(BI.4)$
$\langle$ { BI , SP(p) } ; { BI , SP(p)} ; { SR(r) , (r$\rightarrow p$)}$\rangle$ $(BI.5)$
$\langle$ { BI , SR(r) , SP(p) } ; { BI , SR(r) , SP(p) } ; { (c$\rightarrow p$)}$\rangle$ $(BI.6)$

Productions *(BI.4)*, *(BI.5)*, and *(BI.6)* requires for the style to allow a requestor to be bound to multiple providers and a provider to be bound to multiple requestors. The architecture example presented in figure 1 could be obtained by applying the following productions: BI.1, then BI.3 with r=r1 and p=p1, then BI3 with r=r3 and p=p3, then BI.4 with r=r1 and p=p2, then BI.5 with r=r2 and p=p1, then BI.6 with r=r2 and p=p3 and finally BI.2.

## 4 Orchestrated Interaction (OI) architecture style

The BI architectural style concerns simple interaction model involving a single independent operation on a single service provider. In the case of interactions involving multiple sequences of operations, this is not sufficient any more. Indeed, to manage the workflow of these services we may need to execute invocations in accordance with their ordering constraints or root service requests to the right provider and the service provider response to the right service requestor. Orchestration [6–8] addresses the issue of creating business processes at the message level including business logic, task execution order and transactional aspects. We introduce here, to consider orchestration issue, the orchestrator element and define the orchestrated interaction style.

We present a formal description for this architectural style described by the following graph grammar:
$\langle AX; \{OI\}; \{SR, Orch, SP\}; P_{OI} \rangle$ Where $P_{OI}$ is the set of the following grammar productions:

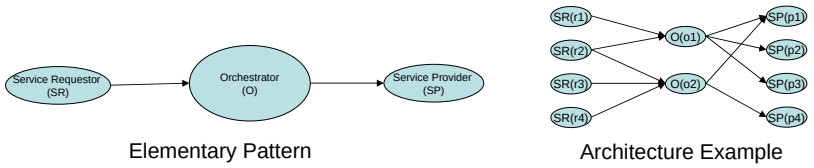


**Fig. 2.** consistent architecture instances of the orchestrated cooperative style

⟨ { AX } ; { } ; { OI } ⟩ (*OI*.1)
⟨ { OI } ; { } ; { } ⟩ (*OI*.2)
⟨ { OI } ; { OI } ; { SR(r) , Orch(o) , SP(p) , (r→o) , (o→p) } ⟩ (*OI*.3)
⟨ { OI , Orch(o) } ; { OI , Orch(o) } ; { SR(r) , (r→o) } ⟩ (*OI*.4)
⟨ { OI , Orch(o) } ; { OI , Orch(o) } ; { SP(p) , (o→p) } ⟩ (*OI*.5)
⟨ { OI , Orch(o) , SR(r) } ; { OI , Orch(o) , SR(r) } ; { (r→o) } ⟩ (*OI*.6)
⟨ { OI , Orch(o) , SP(r) } ; { OI , Orch(o) , SP(r) } ; { (o→p) } ⟩ (*OI*.7)
⟨ { OI , SR(r) } ; {OI , SR(r)} ; { Orch(o) , SP(p) , (r→o) , (o→p) } ⟩ (*OI*.8)
⟨ { OI , SP(p) } ; { OI , SP(p) } ; { SR(r) , Orch(o) , (r→o) , (o→p)} ⟩ (*OI*.9)

Based on the related graph grammar, we can notice that an orchestrator may coordinate several providers (OI.4) and several service requestors (OI.5). We also consider the general case where an orchestrator may share its service requestors and providers with other orchestrators (resp. (OI.6 & OI.8) and (OI.7 & OI.9))[1].

## 5 Composite Architecture styles

The OI architectural style addresses the problem of service interactions that consist of several operation invocations. In this section, we consider the composition issue where the invocation of a service operation involves operations offered by other services. One of the key differences between composition [9–11] and orchestration concepts is related to the fact that composition is concerned by internal implementation of operations [12]. Unlike orchestration protocols which are public documents described by standardized (or nearly) languages such as Business Process Execution Language for Web Services (BPEL4WS) [13], the specification of composite services is generally done within a single company and the composition schema still transparent from the client perspective for privacy considerations. We propose here a composite service style by introducing a new interdependency link between service providers. We distinguish, now, two kinds of links: invocation links (denoted by symbol "→") and composition links (denoted by symbol "⇢"). The last kind of links expresses the composition relation between service providers and their containing composite services.

[1] The architectural style restricting the framework to independent orchestrated patterns, can be obtained by the same graph grammar of the OI style by removing the OI.6 to OI.9 productions.

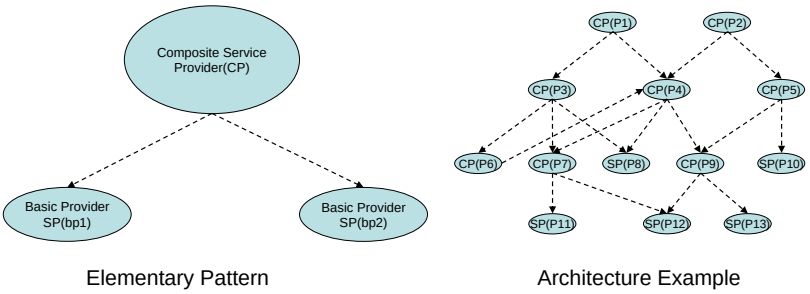


**Fig. 3.** Elementary composite service and a consistent architecture instance of the composite style.

The graph grammar producing patterns of composite service providers can be described as follow: $\langle AX; \{C_{Comp}\}; \{CP, SP\}; P_{Comp}\rangle$ where $P_{Comp}$ is the set of the following grammar productions:

$\langle$ {AX} ; { } ; {Comp}$\rangle$ ($Comp$.1)
$\langle$ {Comp} ; { } ; { } $\rangle$ ($Comp$.2)
$\langle$ {Comp} ; {Comp} ; {SP(p)} $\rangle$ ($Comp$.3)
$\langle$ {Comp} ; {Comp} ; {CP($p_1$) , SP($p_2$) , $p_1$-->p$_2$} $\rangle$ ($Comp$.4)
$\langle$ {Comp , CP($p_1$)} ; {Comp , CP($p_1$)} ; {SP($p_2$) , $p_1$-->p$_2$} $\rangle$ ($Comp$.5)
$\langle$ {Comp , CP($p_1$) , SP($p_2$)} ; {Comp , CP($p_1$) , SP($p_2$)} ; {$p_1$—>p$_2$} $\rangle$ ($Comp$.6)
$\langle$ {Comp , CP($p_1$)} ; {Comp , CP($p_1$)} ; {CP($p_2$) , $p_2$ -->p$_1$} $\rangle$ ($Comp$.7)
$\langle$ {Comp , CP($p_1$) , CP($p_2$)} ; {Comp , CP($p_1$) , CP($p_2$)} ; {$p_2$ -->p$_1$} $\rangle$ ($Comp$.8)
$\langle$ {Comp , SP($p_2$)} ; {Comp , SP($p_2$)} ; {CP($p_1$) , $p_1$-->p$_2$} $\rangle$ ($Comp$.9)

The related graph grammar implies, for composite architectures, that we allow a composite provider to be composed of several atomic providers (Comp.4 & Comp.5) and several composite providers (Comp.7 & Comp.8). It also implies that there may be multiple levels of composition (Comp.7 & Comp.8) but that the lowest level contains exclusively atomic services (because of Comp.3 and Comp.4 and absence of a production of type $\langle$ {Comp , CP($p_1$)} ; {Comp , CP($p_1$)} ; {CP($p_2$) , $p_1$ -->p$_2$} $\rangle$ [2]). An atomic or a composite service may compose different composite services (resp. Comp.6 and Comp.8).

We will extend the styles introduced for the BI and OI architectures to consider the composition issue. According to whether we place the architecture within the framework of a BI or OI architecture, we obtain two architecture styles.

### 5.1 Composite Basic Invocation (CBI) architecture style

The CBI architecture style is described as follows:
$\langle AX; \{CBI\}; \{SR, SP, CP\}; P_{CBI}\rangle$ where $P_{CBI}$ is the set of the following grammar productions:

[2] different from production Comp.7 where we add a composite service of a higher level

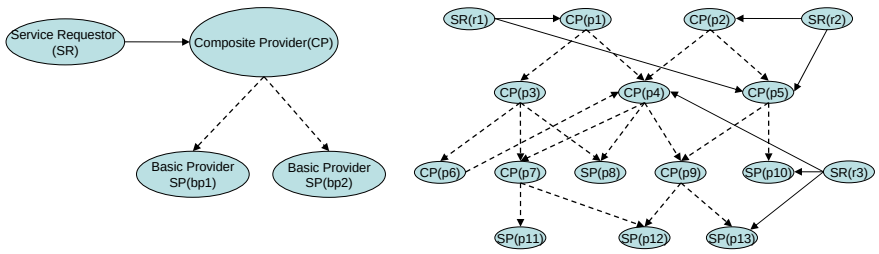


**Fig. 4.** Elementary pattern and consistent architecture of the composite basic interaction style

$\langle$ {AX} ; { } ; {CBI}$\rangle$ $(CBI.1)$
$\langle$ {CBI} ; { } ; { } $\rangle$ $(CBI.2)$
$\langle$ {CBI} ; {CBI} ; {SP(p)} $\rangle$ $(CBI.3)$
$\langle$ {CBI} ; {CBI} ; {CP($p_1$) , SP($p_2$) , $p_1$ --→ $p_2$} $\rangle$ $(CBI.4)$
$\langle$ {CBI , CP($p_1$)} ; {CBI , CP($p_1$)} ; {SP($p_2$) , $p_1$ --→ $p_2$} $\rangle$ $(CBI.5)$
$\langle$ {CBI , CP($p_1$) , SP($p_2$)} ; {CBI , CP($p_1$) , SP($p_2$)} ; {$p_1$ —→ $p_2$} $\rangle$ $(CBI.6)$
$\langle$ {CBI , CP($p_1$)} ; {CBI , CP($p_1$)} ; {CP($p_2$) , $p_2$ --→ $p_1$} $\rangle$ $(CBI.7)$
$\langle$ {CBI , CP($p_1$) , CP($p_2$)} ; {CBI , CP($p_1$) , CP($p_2$)} ; {$p_2$ --→ $p_1$} $\rangle$ $(CBI.8)$
$\langle$ {CBI , SP($p_2$)} ; {CBI , SP($p_2$)} ; {CP($p_1$) , $p_1$ —→ $p_2$} $\rangle$ $(CBI.9)$
$\langle$ {CBI , SP($p$)} ; {CBI , SP($p$)} ; {SR($r$) , $r$→p}$\rangle$ $(CBI.10)$
$\langle$ {CBI , SP($p$) , SR($r$)} ; {CBI , SP($p$) , SR($r$)} ; {$r$→p}$\rangle$ $(CBI.11)$
$\langle$ {CBI , CP($p$)} ; {CBI , CP($p$)} ; {SR($r$) , $r$→p}$\rangle$ $(CBI.12)$
$\langle$ {CBI , CP($p$) , SR($r$)} ; {CBI , CP($p$) , SR($r$)} ; {$r$→p}$\rangle$ $(CBI.13)$

We can notice that the previous grammar contains both the composite grammar ( productions CBI.i for $i \leq 9$ are equivalent to productions Comp.i) and the DI grammar (productions DI.1, DI.2, DI.5, and DI.6 are respectively equivalent to CBI.1, CBI.2, CBI.10, and CBI.11 while DI.3 is equivalent to the composition of productions CBI.3 and CBI.10 and DI.4 is equivalent to the composition of productions CBI.3 and CBI.11.). In addition to the considerations and constraints related to the composite and to the direct interaction styles (which are still valid because of productions inclusion and equivalence), the graph grammar of the CBI style implies that a requestor may interact with a provider at any level of composition (CBI.12 & CBI.13) and even at the lowest one (CBI.10 & CBI.11). It also allow requestors to interact with services belonging to different composition levels (CBI.10 & CBI.11 & CBI.12 & CBI.13).

### 5.2 Composite Orchestrated Invocation (COI) architecture style

The composite orchestrated cooperation architecture style is described as follows:
$\langle AX; \{COI\}; \{SR, Orch, CP, SP\}; (P_{COI})\rangle$ where $P_{COI}$ is the set of the following productions:

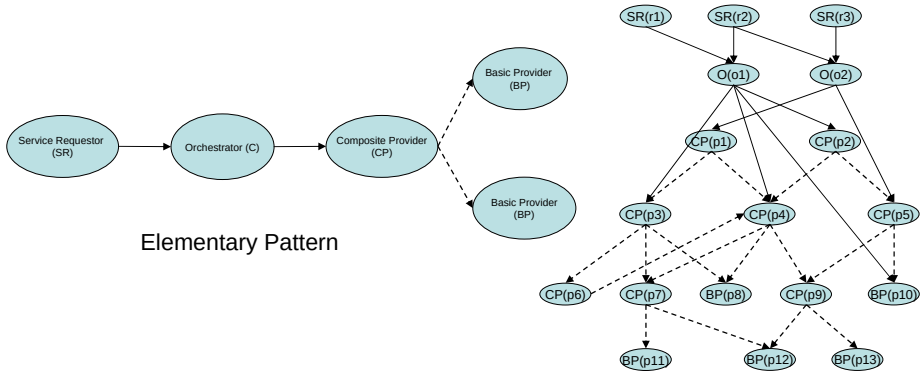


**Fig. 5.** Elementary pattern and an architecture instance of the composite orchestrated interaction style

$\langle$ {AX} ; { } ; {COI}$\rangle$ ($COI$.1)
$\langle$ {COI} ; { } ; { } $\rangle$ ($COI$.2)
$\langle$ {COI} ; {COI} ; {SP(p)} $\rangle$ ($COI$.3)
$\langle$ {COI} ; {COI} ; {CP($p_1$) , SP($p_2$) , $p_1$ ⇢ p$_2$} $\rangle$ ($COI$.4)
$\langle$ {COI , CP($p_1$)} ; {COI , CP($p_1$)} ; {SP($p_2$) , $p_1$⟶p$_2$} $\rangle$ ($COI$.5)
$\langle$ {COI , CP($p_1$) , SP($p_2$)} ; {COI , CP($p_1$) , SP($p_2$)} ; {$p_1$⇢p$_2$} $\rangle$ ($COI$.6)
$\langle$ {COI , CP($p_1$)} ; {COI , CP($p_1$)} ; {CP($p_2$) , $p_2$ ⇢ p$_1$} $\rangle$ ($COI$.7)
$\langle$ {COI , CP($p_1$) , CP($p_2$)} ; {COI , CP($p_1$) , CP($p_2$)} ; {$p_2$ ⇢ p$_1$} $\rangle$ ($COI$.8)
$\langle$ {COI , SP($p_2$)} ; {COI , SP($p_2$)} ; {CP($p_1$) , $p_1$⇢p$_2$} $\rangle$ ($COI$.9)
$\langle$ {COI , SP($p$)} ; {COI , SP($p$)} ; {SR($r$) , Orch(o) , $r$→o,o→p}$\rangle$ ($COI$.10)
$\langle$ {COI , SP($p$) , Orch(o)} ; {CDI , SP($p$) , Orch(o)} ; {$o$→p}$\rangle$ ($COI$.11)
$\langle$ {COI , CP($p$)} ; {COI , CP($p$)} ; {SR($r$) , Orch(o) , $r$→o,o→p}$\rangle$ ($COI$.12)
$\langle$ {COI , CP($p$) , Orch(o)} ; {COI , CP($p$) , Orch(o)} ; {$o$→p}$\rangle$ ($COI$.13)
$\langle$ {COI , orch(o)} ; {COI , Orch(o)} ; {SR($r$) , $r$→o}$\rangle$ ($COI$.14)
$\langle$ {COI , SR($r$) , Orch(o)} ; {COI , SR($r$) , Orch(o)} ; {$r$→o}$\rangle$ ($COI$.15)

Productions from COI.1 to COI.9 are respectively equivalent to productions from Comp.1 to Comp.9 and the considerations and constraints related to the composite style are still consequently valid for the COI style. Considering the COI graph grammar we allow an orchestrator to coordinate several composite services (COI.13), several atomic services (COI.11) and several service requestors (COI.14). An orchestrator may share its atomic providers (COI.11), composite providers (COI.13), and its service requestors (COI.15).

## 6 Conclusion

In this paper, we presented a formal framework for service-oriented architectures description and proposed the graph grammar formalism to specify architectural styles. Elementary styles classification presented here rises from considering two

major concepts which are service composition and orchestration. Thus we described formally the basic interaction style, the orchestrated interaction style, the composite basic interaction style and the composite orchestrated style.

Moreover the proposed approach also makes it possible to check the conformity of an architecture instance to a given style. It is possible to verify if the graph representing a given ad-hoc architecture could be produced from the axiom by generating partially the development tree to overcome scalability problems. We have already experimented such verification approach on client/server architecture.

## References


1. J. Engelfriet and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation*, chapter Node Replacement Graph Grammars, pages 1–94. World Scientific Publishing, 1997.
2. D. Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions On Software Engineering*, 24(7):521–533, July 1998.
3. D. Hirsch, P. Inverardi, and U. Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In $1^{st}$ *Working IFIP Conference on Software Architecture*, pages 127–142, San Antonio, TX, USA, February 1999.
4. H. Fahmy and R. Holt. Using graph rewriting to specify software architectural transformations. In $15^{th}$ *IEEE international Conference on Automated Software Engineering*, pages 187–196, Grenoble, France, September 2000.
5. H. Ehrig, M. Korff, and M. Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In $4^{th}$ *International Workshop on Graph Grammars and Their Application to Computer Science, LNCS 532*, pages 24–37, Bremen, Germany, March 1990. Springer-Verlag.
6. C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, october 2003.
7. J. Estublier and S. Sanlaville. Business processes and workflow coordination of web services. In *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 85–88, Hong Kong, April 2005.
8. T. Lemniotes, G.A. Papadopoulos, and F. Arbab. Coordinating web services using channel based communication. In *Computer Software and Applications Conference COMPSAC*, volume 1, pages 486–491, Hong Kong, September 2004.
9. N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, November-December 2004.
10. S.A. Chun, V. Atluri, and N.R. Adam. Policy-based web service composition. In $14^{th}$ *International Workshop on Research Issues on Data Engineering Web Services for E-Commerce and E-Government Applications RIDE-WS-ECEG'2004*, pages 85–92, Boston, USA, March 2004.
11. J. Liu, J. Cui, and N. Gu. Composing web services dynamically and semantically. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, pages 234–241, Beijing , China, September 2004.
12. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer-Verlag, 2004.
13. *Business Execution Language for Web Services Version 1.1.* http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel/.

# Web Service Conflict Management

Zheng Lu, Shiyan Li, Aditya K. Ghose

Decision Systems Lab, School of IT and Computer Science
University of Wollongong, NSW 2522, Australia
{zl07, sl562, aditya}@uow.edu.au

**Abstract.** Although the development of Web Service technology has made significant progress, there is still a lack of a well-established mechanism for ensuring reliable service composition. Unlike a traditional software module, which runs within a predictable domain, Web Services are autonomous software agents running in a heterogeneous execution environment. Those essential natures of service oriented computing pose challenges to reliable service composition by raising questions such as how to avoid the service conflicts.

## 1 Introduction

Semantic research efforts (OWL-S [8]), formerly known as DAML-S [15], is an upper ontology for services, aimed at achieving the automation of service discovery, invocation, composition and interoperation. OWL-S leverages the rich expressive power of OWL [2] together with its well-defined semantics to provide richer descriptions of Web Services. Recently, semantic web rules language (SWRL) [3, 13] has been proposed to define service process preconditions and effects, process control conditions and their contingent relationships in OWL-S. Though OWL-S is endowed with more expressive power and reasoning options when combined with SWRL, the description provided by a combination of OWL-S and SWRL about service composition is still only a partial picture of the real world. Most of what we know about the world, when formalized, will yield an incomplete theory precisely because we cannot know everything – there are gaps in our knowledge [11]. The ontology of services, on the other hand, is finite and incomplete. Thus, a service composition specified by OWL-S has to deal with partial or incomplete knowledge.

In this paper, we are going to bridge the gap between the semantic service description and multiple operational domains involved by introducing "service assumptions". Currently, OWL-S has no mechanism for handling the explicit description of service assumptions and no method for reasoning about their side-effects. We will extend the current OWL-S and try to define a formal mechanism to reasoning about incomplete knowledge and to address the service conflict issues.

The paper is structured as follows: in Section 2 we explain the examples of Web Service composition problems we address. In Section 3 we explain an atomic service and composite service in general. In Section 4, we define the basic semantics for planning-based service composition domain. In Section 5, we present a framework for

reasoning about incomplete knowledge in service composition context. Finally, in Section 6, we present related work and our conclusions respectively.

## 2 Motivating Examples

The following example of a travel agency is used to explain the service conflicts which may be caused by incompleteness of information during the dynamic service composition. Our example uses the often presented travel agency service package. A typical use case could be the arranging of a trip with a hotel booking together with a car rental and a sightseeing service. To simplify this use case, we assume this composite service is executed in sequential manner (i.e. hotel booking service, then car rental service, finally sightseeing service). Assume that when requesting this composite travel agency service, user specifies his preferred car model, for example, a city car. Obviously, this car will be used for sightseeing which is generated from this composite service. If the functionality matches user's requirement, then car rental service is invoked. In the real world, most likely, the car rental service providers have some service policy about usage of rental cars. However, when the car rental service is invoked, we don't have any information about what kinds of sightseeing plan generated from the execution of the service, in other words, we don't know how the rented car will be used. The point here is that different sightseeing plans may be associated with different roads, and it is may not be allowed for a rented car to drive on certain roads. For example, the desert dune exploration plan is dynamically generated from the service and a city car is used for the desert dune exploration. Certainly it is not ideal situation for both the car rental company and the customer.

Thus, to ensure integrity of service composition, there should be a mechanism to deal with incompleteness of information during the dynamic service composition. The solution to this problem is to use the service assumptions. In this example, to prohibit the illegal usage of the rental car, the car rental service could make the assumption that "city cars do not drive on dune, beach, unsealed road…". If the contrary evidence appears (e.g. a dune exploration) from the succeeding service executions, then we can conclude that there is a violation to the car usage policy. In other words, if we can get additional information which is explicitly contradictory to the service assumptions in context of the service composition, and then the potential service conflicts are detected. The inability to make assumptions therefore translates into an inability to deal with exceptions [10]. Before formally introducing our framework in next Section, we will clarify some basic definitions about the Web Services.

## 3 Atomic Service and Service Selection

### 3.1 Atomic Service

An atomic service only performs a single function, each atomic service *ws* is described by a tuple $ws =< p, e, a >$, where $p$ represents service precondition which must be true

for the service $ws$ to be invoked, $e$ represents the change of world state i.e. the effects after service $ws$ completes and $a$ represents the service assumptions. Note that $p$ and $a$ are different. It must be possible to establish that $p$ is true for $ws$ to be invoked. On the other hand, we only need to establish that $a$ is consistent with what is known i.e. nothing is known that contradicts $a$. $p$ is the strong condition which must be true in order to execute the service $ws$, while $a$ is a weak condition. Initially we assume $a$ to be true, unless we get additional information which is explicitly contradictory to $a$.

### 3.2 Service Selection

The automated process of Web service compositions over that of software component compositions holds some additional critical issues, such as service matching, selection and retrieval. UDDI [5] provides a mechanism for the Web wide service registry, in which descriptions of Web Service in UDDI are stored and searched by Category. OWL-S allows us to semantically describe the capabilities of Web Services, thus it is possible to perform logic inference for the service matching. [6] Provides one way to combine these two efforts, by which services defined in OWL-S can be registered with UDDI and allows UDDI engines to exploit OWL-S semantic information to facilitate the retrieval of Web Services. In this proposed framework, let

1. $ws_i$ represents an atomic service
2. $WS$ is the set of all Web Services, $ws_i \in WS$
3. All Web Service descriptions are held in their corresponding categories $\{cat_1, cat_2 ... cat_n\}$ $cat_i$ is a tangible areas split from the service registry, for example downloadable Multimedia.
4. $CAT$ is the set of all service categories $cat_i \in CAT$ , $cat_i \in WS$ , $cat_i = \{ws_1 ... ws_m\}$
5. Service selection function $sel : CAT \rightarrow WS$ which takes a certain service category as its input and give us an atomic service based on the service matching i.e. $sel(cat_i) = ws$.

Every atomic service $ws$ in the rest of this paper refers to the Web Service which is produced by the service selection defined above. For more details about the service matching, interested readers may refer to [6, 7].

### 3.3 Composite Service

Intuitively, a composite Web Service which performs combined functions may include multiple atomic services. A composite service is the combination of the multiple atomic services $ws_i$, where 0 < i < n, a composite service $CompWS$ can be represented as: $CompWS = \{sel(cat_1) ... sel(cat_n)\}$

Because participants of the service composition do not necessarily share the same objectives and background, without reasoning about incomplete knowledge and its side

effects during the service execution, conflicts are easily accommodated in the service composition context.

## 4 Service Composition as Planning

It is often assumed that a business process or application is associated with some explicit business goal definition that can guide a planning-based composition tool to select the right service [12]. Typically, classical planners presuppose complete and correct information about the world. However, this simplifying assumption is not suitable and unrealistic in the context of Web Service composition. Each node of service composition is designed, owned, or operated by distinct parties, thus the agent may not have complete information about world. To reasoning about incompleteness of information in the service composition context, we extend current semantic Web Service description OWL-S by introducing service assumptions. Assumptions, in this framework, together with states, preconditions, effects, and goals are all specified in Description Logic $L$ [1].

Now we are prepared to define the semantics of a service composition domain. A state $S$ is a snapshot which describes the world with respect to the service composition context. The state $S$ in this work is extensionally defined as a set of positive or negative ground atomic formulas (atoms). These atoms which may change their values during the state transition are called *fluent*, while for those which don't change are called *state invariant*. Unlike traditional planning, here $S$ is a partial description of the world. A state transition $t$ is represented as a tuple $t =< s, ws, s' >$, where $s, s'$ are states and $ws$ is an atomic service. It is also worthwhile to mention that the initial state $s_0$, in this framework, is a partial description about the world. A goal $G$ is set of conjunctions of atoms which need to hold in a desired world state or say final state. A service composition plan for a goal $G$ is a sequence of state transitions of atomic services, and the transitions lead from an initial state to a final state where all ground atomic formulas in the goal are true.

In the process of the state transition, there are three types of knowledge about the current world which represent the state transition. Let $SEN_i$ denote a set of sentences used to change the state $S_i$. This set of sentences can be partitioned into three categories, namely state invariants, expansion and update, which is defined as: $SEN_i = \{Inv_i \mid Exp_i \mid Upd_i\}$. In rest of the paper, we will use symbol |= to represent logical entail.

1. State invariant $Inv_i$ denotes a set of sentences which can be entailed by the knowledge in the previous state, defined as: $S_{i-1} \models Inv_i$
2. State expansion $Exp_i$ denotes a set of sentences which cannot be entailed by the knowledge in the previous state and its negation also cannot be entailed by the knowledge in the previous state, defined as: $S_{i-1} \not\models Exp_i$ and $S_{i-1} \not\models \neg Exp_i$
3. State update $Upd_i$ denotes a set of sentences whose negation can be entailed by the knowledge in the previous state, defined as $S_{i-1} \models \neg Upd_i$

In our framework, for any atomic service $ws$ , Let $WS$ be the set of all Web services, $E$ is the set of all service effects, $P$ is the set of all service preconditions, we define the following extraction functions:

1. Effect extraction function $f_e : WS \to E$ which takes an arbitrary atomic service $ws_i$ as an input, and extracts the effect $e_i$ of $ws_i$ as its output. $e_i$ is a set of primitive effects of $ws_i$ and every primitive effect is a partition with the state invariant, expansion and update i.e. $f_e(ws_i) = e_i$ and $e_i = \{eInv_i \mid eExp_i \mid eUpd_i\}$ in which $eInv_i, eExp_i, eUpd_i$ denote state invariant, expansion and update respectively.

2. Precondition extraction function $f_p : WS \to P$ which takes arbitrary atomic service $ws_i$ as an input, and extracts the precondition $p_i$ of $ws_i$ as its output i.e. $f_p(ws_i) = p_i$ . Similar to the effect extraction function: $p_i = \{pInv_i \mid pExp_i \mid pUpd_i\}$ .

Following the definitions above, we can define the generic state transition operator as: $S_i = \Delta(pUpd_i, eUpd_i, S_{i-1}) + eExp_i + pExp_i$

which means the state transition from $S_{i-1}$ to $S_i$ is completed by means of applying $pUpd_i$ and $eUpd_i$ to the state $S_{i-1}$ orderly, then add the two types of expansion of knowledge ( $eExp_i$, $pExp_i$ ) to the state $S_{i-1}$ . Note that the order of applying state update must be strictly followed.

## 5 Defeasible Reasoning in Service Composition

Comparing with the traditional software development, a dynamic service composition is an automated process with less human intervention. Usually, it doesn't have a predefined boundary, based on which the problems of uncertainty and incompleteness of information could be tackled. Unpredictable service executions and a dynamic changing context complicate dynamic service composition in many ways. Inspired by Non-monotonic logic [4], the following sub-section will attempt to provide a formal framework for reasoning about incomplete knowledge in service composition context.

### 5.1 Defeasible Reasoning Framework

In this work, our conflict checking and reasoning about incompleteness of information work with three kinds of rules, namely absolute rules, defeasible rules and defeaters [4]. The absolute rules which are interpreted in the classical sense means whenever the premises are indisputable then so is the conclusion. On the other hand, a defeasible rule is that whose conclusion is normally true when its premises are, but certain conclusions may be defeated in the face of new information. Defeasible rules can be defeated by contrary evidence or by defeaters. Defeaters represent knowledge which is to prevent defeasible inference from taking place. We use the operator $\Rightarrow$ for absolute rules, ~> for defeasible rules and $\mapsto$ for defeaters.

Let $ws_i$ represents a Web Service which is produced by the service selection function (see section3.2), $a_i$ represents the assumptions of $ws_i$ , $p_i$ represents the precondition

of $ws_i$, $e_i$ represents the effects of $ws_i$ and $isValid(ws_i)$ represents a Web Service whose preconditions can be satisfied. Based on Nute's defeasible reasoning [4],

Absolute rule: $p_i \Rightarrow isValid(ws_i)$ (Rule A)

Which means only precondition holds, and then the service is a valid candidate service to participate service composition.

Defeasible rule: $isValid(ws_i) \sim> e_i$ (Rule B)

Which means normally $e_i$ is the conclusion of $isValid(ws_i)$, but which $e_i$ may be defeated in the face of new information.

Defeater: $\neg a_i \mapsto \neg e_i$ (Rule C)

Which means given an assumption $a_i$, if the negation of the assumption is entailed by a given state of knowledge, it will prevent the Rule B from making the conclusion $e_i$.

### 5.2 Outdated Assumptions and Assumption Database

If the negation of all sentences in $e_i$ is entailed by some states $s_j$, where $e_i$ is an effect of Web Service $ws_i$ and j > i, then the assumption $a_i$ associated with $e_i$ called the outdated assumption. Formally, the outdated assumption can be represented as: $\forall x \in e_i \ \exists j > i \ such \ that \ \neg x \in Cn(s_j)$, Where $Cn(s_j)$ denotes logical closure of $S_j$.

The outdated assumptions are not allowed to participate defeasible reasoning. A simple example of an outdated assumption is: a book borrowing service assumes that the borrower is in same city as the library. When the borrowed book is returned, we say this assumption is outdated.

To conduct the defeasible reasoning about the current state of the world, it is necessary to describe and record various assumptions generated during the service composition planning. In this framework, we maintain an assumption database $D_\alpha$ to store these assumptions and their relevant effects as a pair $< a_i : e_i >$. Same as preconditions and effects, assumptions are represented as ground literals.

### 5.3 Defeasible Reasoning Process

In this subsection, we are prepared to illustrate the process of constructing the service composition plan based on our proposed framework. Service composition planning can be viewed as a process of resolving conflicts and gradually refining a partially specified plan, until it is transformed into a complete plan that satisfies the goal.

Service Composition planning is similar to the classical planning [9] in that each world state is represented by a conjunction of literals and each Web Service is related to a transition between those states. However, unlike classical AI planning techniques, in this proposed framework, the planner is the rule based system which allows making tentative conclusions and revising them in the face of additional information. In other words, the planner is endowed with the ability to reasoning about incomplete information in the service composition context.

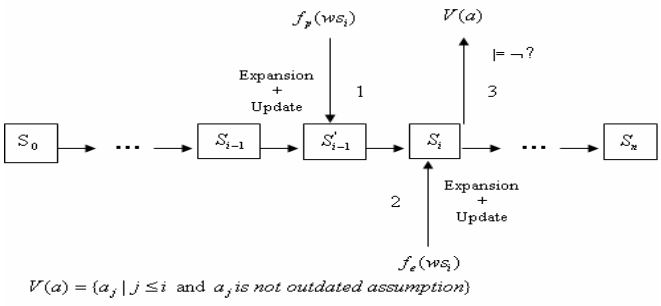


**Fig. 1.** Defeasible Reasoning Process

For any state $S_{i-1}$, Web service $ws_i$ is not applicable to the state until certain minimal criteria are met. $ws_i$ is specified in terms of the precondition $p_i$, effect $e_i$ and assumption $a_i$, where $p_i$ must be satisfied for $ws_i$ to be valid (Rule A), the effect may be concluded (Rule B) and the negation of the $a_i$ plays the role in being the defeaters(Rule C).

A state in our framework is not a complete view of the world. Usually, an agent is forced to perform sensing operations which is aiming at finding out the information which could satisfy the precondition $p_i$. Like "1" showed in the Fig. 1, the sensing operation may lead to knowledge expansion and update of the state $S_{i-1}$. When the sensing operations complete, if $p_i$ is satisfied, we can conclude that $ws_i$ is applicable to the current state (Rule A). Due to the expansion and update of knowledge to state $S_{i-1}$, before the transition to state $S_i$, we get an intermediate state $S'_{i-1}$ which holds the current knowledge of the world after the agent's sensing operation. Following the sensing operations, effect $e_i$ is applied to the current world state $S'_{i-1}$ to simulate the action. Again, the effect $e_i$ may expand and update the knowledge of the current state (Rule B). This process can be presented as generic state transition operation as we defined in Section 4.

One of the main features in this proposed framework is the ability to describe various service assumptions and support defeasible reasoning with these assumptions. Assumptions generated from the service composition planning are represented as a set of ground literals stored in the assumption database $D_\alpha$. After the effect is applied to the current state, the knowledge in the state may be expanded or updated. For the new state of knowledge, the planner needs to carefully perform the checking to see whether any outdated assumption is in $D_\alpha$. Because the outdated assumptions are not allowed to participate the defeasible reasoning, all outdated assumptions are deleted from $D_\alpha$. Next, it is to find the defeaters by the mean of checking whether any negation of assumptions can be entailed by the current state. The negation of service assumption which is not outdated plays the role in being a defeater, which prevents the effects associated with this assumption being applied to the state (Rule C). Up to now, the

process of state transition from $S_{i-1}$ to $S_i$ is completed. We have illustrated that how the new world state is reached in the presence of possibly conflicting rules.

## 6 Summary

The capability of a Web Service is specified by WSMO[14] in terms of precondition, postcondition, assumption, effect and some others properties. However, the assumption we proposed in this work is different from the assumption in WSMO. The assumption proposed in this work extends the semantics of OWL-S for the purpose of explicitly supporting defeasible reasoning and trying to tackle the problem of incomplete information in service composition context.

The goal of dealing with incomplete information in the service composition context is certainly a challenging task. The proposed framework is an attempt at tackling the problem of how to achieve consistent service composition when information available is insufficient.

In this work, we have extended the OWL-S to a richer service description representation schema by introducing the service assumptions. We also adopted defeasible rules for reasoning with various assumptions. We illustrated how knowledge based planning could reason about incomplete knowledge in service composition context and construct the service composition plan. During the process of the service composition we showed that absolute rules could be used for service precondition satisfaction, especially defeasible rules and defeaters could be employed to make tentative conclusions based on the available information, and to detect potential conflicts in service composition when further suitable information about the problem is available.

## References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., Eds. 2003. The Description Logic Handbook:Theory, Implementation and Applications. Cambridge University Press.
2. Dean, M. and Schreiber G. OWL Web Ontology Language Reference W3C Recommendation, http://www.w3.org/tr/owl-ref/. Feb 2004.
3. Benjamin N. Grosof, Ian Horrocks Description Logic Programs: Combining Logic Programs with Description Logic ACM 1581136803/03/0005.
4. Donald Nute. Defeasible logic. In D. Gabbay and C. Hogger (eds.), Handbook of Logic for Artificial Intelligence and Logic Programming, Vol. III, Oxford University Press, 1994:353-395.
5. Kreger, H. Web services Conceptual Architecture (WSCA 1.0). http://www-4.ibm.com/software/solutions/Webservices/ , 2001
6. M. Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara. "Importing the Semantic Web in UDDI". In *Proceedings of Web Services, E-business and Semantic Web Workshop*.
7. M. Paolucci, T. Kawmura, T. Payne and K. Sycara. Semantic Matching of Web Services Capabilities. In First Int. Semantic Web Conf., 2002
8. OWL-S White Paper: OWL Services Coalition. OWL-S: Semantic markup for Web services, 2005. http://www.daml.org/services/owl-s/1.1/overview/#1

9. R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 88–97. Kaufmann, San Mateo, CA, 1990.
10. R. V. Guha. Contexts: A Formalization and Some Applications. PhD thesis, Stanford University, 1991.
11. Reiter R. A logic for default reasoning, Artif Intell 1980; 13:81–132.
12. S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April 2002.
13. "Semantic Web Rule Language", May 21, 2004. http://www.w3.org/Submission/2004/03/.
14. WSMO working group. WSMO homepage, since 2004. http://www.wsmo.org/.
15. Ankolenkar et al., 2001 A. Ankolenkar, M. Burstein, T. Cao Son, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S.McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng: DAML-S: Semantic Markup For Web Services, http://www.daml.org/services/daml-s/2001/10/daml-s.html.

# An Engineering Method for Semantic Service Applications

Guido Laures and Harald Meyer and Martin Breest

Hasso-Plattner-Institute for IT-Systems-Engineering
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
{guido.laures|harald.meyer|martin.breest}@hpi.uni-potsdam.de

**Abstract.** Industrial usage scenarios for semantic services are currently hard to find. The lack of standardized ontologies for certain business domains, the complexity of the technologies, and very few concrete design methodologies hamper the acceptance of semantic services-oriented approaches.
The service engineering method presented in this paper demonstrates how end-user centric applications can benefit from the potential of semantic services and their (semi-)automated composition. A practical implementation example taken from the telecommunication sector proves the applicability of the presented approach.

## 1 Introduction

Developing applications that re-use and compose the functionality provided by existing services reduces development costs and is therefore one of the key engineering goals. A *composite application* provides its specific functionalities by invoking existing services or compositions of them. However, the direct addressing of services inside a composite application using standard web technologies implies that it will be statically bound to external functionality. This leads to the following issues:

- The integration of new services always implies a change in service binding inside the application, which in most cases induces additional coding.
- The internal application flow can hardly be changed without rebuilding the application.
- The reliability of the application depends on the availability and reliability of external services that in most cases are out of the control of the developing organization of the composite application.

Throughout this paper we will use the following definitions: a service is synonymous to a web service as defined by the Web Service Architecture [1]. A service composition is a process consisting of service interactions to reach a defined goal [2]. A semantic service is a service for which a machine-processable semantic description of its functionality exists.

## 2 Related Work

As shown in [3–6] the issues discussed can be tackled using semantic services. While they allow for the development of more flexible, adaptable, and fault-tolerant systems, their application to industrial scenarios is lacking. Several problems hamper their acceptance. Industry-strength, reusable ontologies are missing. Required technologies are still immature and hard to use. The missing link, however, is a concrete design methodology that guides the design of semantic service-oriented architectures. In this paper we present core elements of such a methodology and elaborate its most important activities.

Our methodology is not self-contained. It is rather meant to be integrated into existing engineering processes. The development of a full featured methodology is out of the scope of this paper. As most parts of a software engineering process are common ground, it would not be very helpful either. Instead we are restricting ourselves to the design of the semantic service-oriented part. Combined with existing development methodologies (e.g. UP [7], V-Modell XT [8], OOSE [9]), our approach can overcome today's limitations. We will focus on the specification of the core activities to be conducted in a semantic service application engineering process. Unlike others [10] the proposed methodology is independent of specific semantic service specification languages. Instead, it is described on a conceptual level claiming to be applicable to various semantic service languages and architectures.

## 3 Requirements on a Semantic Service-Oriented Application Design Methodology

We demonstrate our methodology using a mobile scenario. The scenario aims at the development of an attraction booking application enabling end users to use their mobile devices to gain information about attractions, to book tickets, and to get a route to the selected attraction. There are three stakeholders in this scenario: The *end user* accesses the attraction booking application using a mobile device. The *end service provider* is responsible for developing and running the composite attraction booking application remotely accessed via a mobile device. The providers of external services that are invoked by the attraction booking application are called *external service providers*. Our methodology aims at the end service provider.

In the following, we will examine the requirements of a methodology tailored for semantic service-oriented applications. The usage scenario following the sketched method will prove its applicability. In the last section we provide a conclusion and an outlook how the methodology outlined in this paper can be enhanced and detailed in the future.

There are two alternate approaches to develop a semantic service. The first one is to develop a service from scratch and provide a semantic description using a semantic service specification language (e.g. WSML [11]). The other way is to develop only the semantic specification for an existing web service and

attaching the semantic specification to its WSDL. In this paper we develop a methodology for the second strategy, as this approach only adds few additional requirements to any engineering methodology already applied in developing the core functionality, thus easing the barrier to adoption.

*Requirement 1: Guide the design decisions on service granularity* The granularity and also the used ontology concepts of a semantic service specification determine the flexibility and adaptability of the composite application using the specified service. If a semantic specification uses very fine granular concepts they are likely to be composed and re-used in several use cases of the application. But, too fine-granular descriptions cause a complex ontology and high efforts to develop the specification itself. A more coarse-grained specification on the other side is easy to develop but might on the other hand avoid the usage of the service in a significant number of service compositions of the application. Thus, a methodology for the development of composed semantic service applications needs to provide guidance for the selection of the most suitable concept granularity.

*Requirement 2: Provide a concise terminology to support communication and collaboration of stakeholders from different domains* Services defined and used in a SOA ideally correspond to concrete business functionality. This implies that projects dealing with the integration of services depend on an effective information exchange between business and technology experts. Therefore, it is important to define a clear terminology that all roles involved in the project agree on. Such a terminology provides definitions for concepts from the SOA domain as well as refined stakeholder role specifications. SOA projects are integration-driven making a stakeholder analysis more complicated as compared to classical application developments with a clearer distinction of the roles involved.

*Requirement 3: Provide a trace from business models to machine-processable domain ontologies* Domain models elaborated using a traditional software engineering process tend to serve as a documentation only. Thus, they are not likely to be transferred into tangible artifacts processable by computational resources of the system under development (SUD). In semantic service based applications it is crucial to follow a methodology that provides traceability from the domain model to a machine-processable ontology. As ontologies form the basis for a number of semantic services re-used in different applications, the methodology also needs to provide guidelines for the integration as well as the enhancement of existing ontologies.

## 4 Semantic Service-Oriented Application Engineering Methodology

For each activity of the methodology presented in the following section we will use a common description pattern containing the following elements:

*Description:* a textual description of what is to be conducted in the activity.

*Responsible Role:* the role in the engineering process responsible for the conduction of the activity

*Input Artifacts:* the artifacts of the engineering process needed as an input to this activity

*Output Artifact:* the generated artifacts of this activity

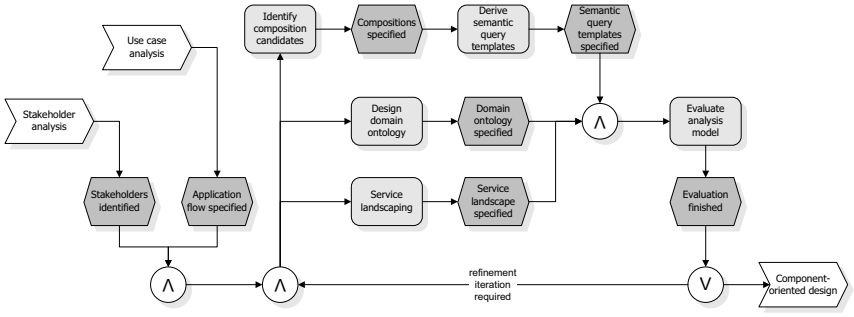


**Fig. 1.** EPC of the sketched methodology

Figure 1 presents an EPC for the core activities of the proposed methodology. In an iterative approach the presented activities aim at bridging the gap between the use case analysis and the design phase taken from a classical development process.

### 4.1 Identify Service Compositions

The application flow is a top-level view of the flow inside the application. Required functionality and their interactions are identified. In this early stage of development it is unclear how this functionality maps to the landscape of available or planned services. Based on the application flow, the use cases of the application that will be implemented with the help of compositions of external services are identified. Figure 2 shows the application flow of the Attraction Booking composite application. The application flow includes the three main use cases: find attractions, book attraction, and get a route description. Each of them is realized using service compositions. As the services used are subject to frequent changes, these parts of the application flow are ideal points for automatically created service compositions.

*Responsible Role* Application Designer.

*Input Artifacts* To identify the service compositions an application flow specification is mandatory. Furthermore, a rough idea of the available service landscape is required.

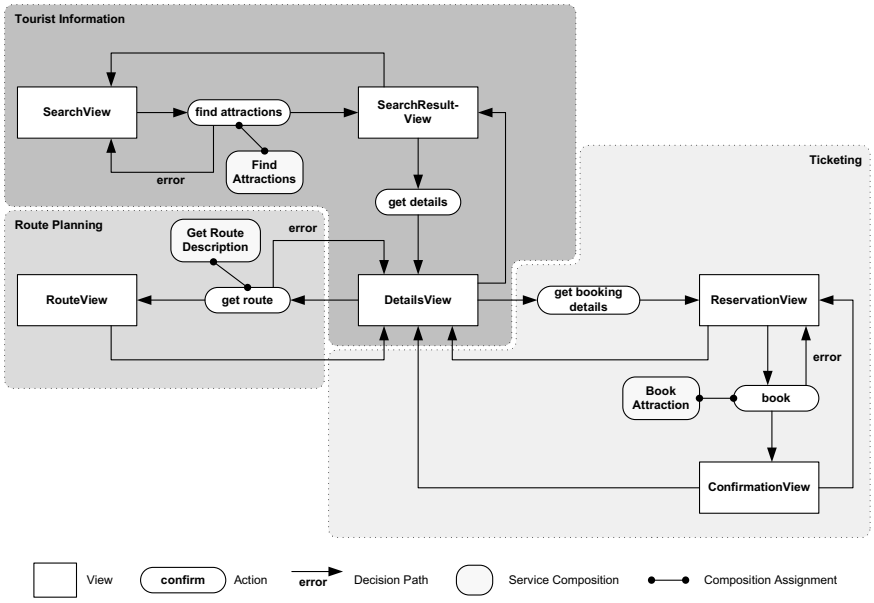


**Fig. 2.** Identified Service Compositions

*Output Artifacts* The identified service compositions and their context inside the application flow.

### 4.2 Design Domain Ontology

Several approaches to the design or engineering of ontologies exist. This does not only include the design from scratch (e.g. [12–14]) but also reusing ontologies [15]. As our methodology is agnostic to the ontology design approach used and we do not intend to develop our own approach, we will not further elaborate this activity in this paper.

*Responsible Role* Domain expert and system analyst.

*Input Artifacts* The application flow specification as well as the composition identification is needed to identify the concepts of the to be developed ontology.

*Output Artifacts* The output is a ontology containing all concepts needed for the semantic service specification of the services in the service landscape that will be elaborated in the Service Landscaping action described below.

### 4.3 Service Landscaping

A crucial part of every service-oriented project is the identification and analysis of existing and newly developed functionality to be integrated in the SUD as services. We call this action service landscaping. To conduct service landscaping it is important to have the domain ontology in mind elaborated as output artifacts of the former actions of the methodology. Our methodology does not rely on automated service discovery using web search as proposed in [16]. We are convinced that in real business-related scenarios the selection of integrated basic services and their providers needs to be a manual process. Therefore, the service landscaping in our methodology mainly relies on a selection of basic web services identified by a manual search and identification process. The specifications of the external services (WSDL) need to be manually mapped to the concepts of the domain ontology afterwards.

*Responsible Role* Architect together with domain experts.

*Input Artifacts* The composition identification is the most important input of this activity. However, it is important that a redesign of the compositions might be necessary if no adequate external basic services can be identified during this action. Another input artifact is the concept specifications of the domain ontology.

*Output Artifacts* The output of this activity is a set of semantic service specifications and their mapping onto the grounded external services.

### 4.4 Derive Semantic Query Templates

After a successful service landscaping, all required functionality can be mapped to services. Services and compositions of these services embedded in the application flow have been identified. Nevertheless, the system is still unusable for end users. To request functionality, semantic queries are issued. In response, service compositions are created and enacted. End-users are not able to specify queries directly as they usually include complex logical expressions and require basic programming skills.

Our approach therefore includes the specification of templates for semantic queries. Instead of requiring end-users to specify semantic queries, they are merely asked for data input and additional properties. In turn, the user-specified properties serve as values for the variables of the query templates (e.g. as it is often done in SQL). These include boolean decisions that trigger the inclusion of logical expression in the semantic query. More complex scenarios are possible as well: lists allow the user to select one of several possibilities.

While this approach reduces the flexibility of automated service composition, it results in a better usability of the end-user application. As semantic query templates are known, the system as a whole is easier to use and more reliable than a system where no restrictions on queries are in place. As several distinct user inputs may be linked together, a huge amount of instantiations for each template

is possible. As composition is performed during run-time, most adaptations are still possible, even in case of registration or de-registration of services.

*Responsible Role* Application designer.

*Input Artifacts* The input for the design of semantic query templates includes the identified service compositions and the ontology. For each possible composition one template is created according to the ontology.

*Output Artifacts* The output are the semantic query templates for each composition and their integration into the user interaction.

## 5 Practical Applicability

The attraction information scenario we developed using the described methodology is based on the ASG reference architecture implementation. The service landscaping showed adequate public services needed for the realization of such a scenario still to be lacking. We therefore were forced to define and use custom web services as simulation. Because of the lack of a fully-featured reasoning engine for WSML-based services we had to manually define the ontology as well as the service semantics using a frame logic implementation (Flora-2). Therefore, the whole ontology definition based directly on Flora expressions proved to be hard to understand for non-technical people. Thus, the traceability from business models to machine-processable ontologies is limited due to missing tools able to provide a better abstraction of the underlying technology.

## 6 Conclusion and Outlook

In this paper we sketched the basic elements of an engineering methodology for semantic service based applications. The presented approach focuses on applicability and has been validated by a prototypical implementation using the ASG reference architecture implementation. Even though in principal it is possible to start developing composite application using semantic services we found out that tool support as well as the availability of external web services to be used in service compositions is still far from being production-ready. More scientific efforts need to be spent especially in the language and tool support for the reasoning of semantic services. Furthermore, the development of business models for the provision and integration of external web services (semantic and non-semantic) is crucial for the full exploitation of the potential of web service technologies in industries.

## References

1. Booth, D., et.al.: Web Services Architecture, W3C. (2004) http://www.w3.org/TR/ws-arch/.

2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer (2003)
3. Fensel, D., Bussler, C.: Semantic web services. IEEE Intelligent Systems **16** (2001) 46–53
4. Fensel, D., Bussler, C.: The web service modeling framework (wsmf). Electronic Commerce: Research and Applications **1** (2001) 113 – 137
5. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web services standards. In: Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada, USA (2003) 395 – 401
6. Martin, D., et al.: Bringing semantics to web services: The owl-s approach. In: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA (2004)
7. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley (1999)
8. Bundesrepublik Deutschland Germany: V-Modell XT. (2004) http://h90761.serverkompetenz.net/v-modell-xt/Release-1.1/Dokumentation/html/.
9. Oesterreich, B.: Objekt-orientierte Softwareentwicklung. Oldenbourg (2001)
10. Jaeger, M.C., Engel, L., Geihs, K.: A methodology for developing owl-s descriptions. In: In First International Conference on Interoperability of Enterprise Software and Applications Workshop on Web Services and Interoperability, Switzerland (2005)
11. de Bruijn, J., et.al.: The Web Service Modeling Language WSML, DERI. (2005) http://www.wsmo.org/TR/d16/d16.1/v0.2/20050320/.
12. Grüninger, M., Fox, M.S.: Methodology for the design and evaluation of ontologies. In: Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95, Montreal, Canada (1995)
13. Uschold, M., King, M.: Towards a methodology for building ontologies. In: Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95, Montreal, Canada (1995)
14. Staab, S., Studer, R., Schnurr, H.P., Sure, Y.: Knowledge processes and ontologies. IEEE Intelligent Systems **16** (2001) 26–34
15. Studer, R., Benjamins, R., Fensel, D.: Knowledge engineering: Principles and methods. Data and Knowledge Engineering **25** (1998) 161 – 197
16. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web, London, UK, Springer-Verlag (2002) 333–347

# Author Index