ACT4SOC 2007

Marten van Sinderen (Ed.)

Architectures, Concepts and Technologies for Service Oriented Computing

Proceedings of the
1st International Workshop on
Architectures, Concepts and Technologies
for Service Oriented Computing -ACT4SOC 2007

INSTICC PRESS

In conjunction with ICSOFT 2007
Barcelona - Spain, July 2007

Marten J. van Sinderen

Architectures, Concepts and Technologies for Service Oriented Computing

Proceedings of the
1st International Workshop on
Architectures, Concepts and
Technologies for Service Oriented Computing
ACT4SOC 2007

In conjunction with ICSOFT 2007
Barcelona, Spain, July 2007

INSTICC PRESS
Portugal

Volume Editor

Marten J. van Sinderen
University of Twente
Enschede, The Netherlands

1st International Workshop on
Architectures, Concepts and
Technologies for Service Oriented Computing
Barcelona, Spain, July 2007
Marten J. van Sinderen (Ed.)

Copyright © 2007
INSTICC PRESS
All rights reserved

Printed in Portugal

ISBN: 978-989-8111-08-1
Depósito Legal: 261300/07

Foreword

This volume contains the proceedings of the First International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2007), held on July 22 in Barcelona, Spain, in conjunction with the Second International Conference on Software and Data Technologies (ICSOFT 2007).

The ACT4SOC workshop aims at serving as a forum for researchers and practitioners, from academia and industry, to meet and to discuss the goals, benefits, achievements and challenges of Service Oriented Computing (SOC). SOC has emerged as a helpful paradigm for designing, building and using IT solutions, based on a set of architectural guidelines and concepts referred to as the Service Oriented Architecture (SOA).

SOC/SOA can be seen as a next step in the evolution of modular middleware approaches, such as CORBA, DCOM and J2EE. The objective of SOA is to make the development of IT solutions for end-users easier and more cost-effective through the adoption of service orientation. Service orientation has service as its central concept to denote a function independent of its possible implementations and independent of its possible user environments. Moreover, service orientation entails the existence of a distributed computing platform, which supports registration of services, discovery of services, invocation of services, and coordination of services. Hence, this foundation, when underlain with proper technology, contributes to important business objectives, including 'plug-and-play' interoperability through 'loose coupling' and 'technology transparency'.

Although SOA is independent of any specific technology or technology platform, its benefits can only be achieved through support from concrete technology. The current technology of choice for realizing SOA is Web services, which comprise practical base standards for service orientation, while building on the popularity of Web technology and ubiquitous Internet standards.

The uptake of Web services based SOC is impressive, but Web services still have important limitations, and they are by no way a complete and satisfactory realization of the distributed computing platform for SOC. Many practical and fundamental challenges remain and need to be addressed in order to achieve the full potential of SOC.

Among these are quality-of-service, security, semantic interoperability, automatic composition, and business-technology alignment.

The goal of this workshop is to focus on the fundamental challenges related to SOC, to discuss what architectural/conceptual foundation is needed, and how this foundation can be supported by new or (extensions of) available technology.

Following the ACT4SOC 2007 Call for Papers, 23 paper submissions were received, from which 8 papers were selected for a 30-minutes oral presentation during the workshop and for publication in this proceedings. Each of the submitted papers went through a thorough review process, with at least 3 reviews per paper. Due to the number of reviews and the professionalism of the authors, program committee members and reviewers, I am confident that all selected papers are of high quality. The selected papers are also a good illustration of different topics that are currently under research. They have been grouped in three presentation sessions during the workshop, named "Architectures", "Concepts" and "Technologies".

I like to take this opportunity to express my gratitude to all people who contributed to ACT4SOC 2007. My thanks go to the authors, who provided the main content for this workshop, and to the program committee members and reviewers, who provided constructive comments that contributed to the quality of the content. I also like to thank the ICSOFT secretariat, especially Mónica Saramago, for the excellent organizational support. Finally, I appreciate the opportunity given by the ICSOFT chair, Joaquim Filipe, to organize this workshop in conjunction with ICSOFT 2007.

I wish all presenters and attendees an interesting and productive workshop, and a pleasant stay in the beautiful and exciting city of Barcelona.

July 2007

Marten van Sinderen

University of Twente, Enschede, The Netherlands

Workshop Chair

Marten J. van Sinderen
University of Twente
Enschede, The Netherlands

Program Committee

Marco Aiello, University of Groningen, The Netherlands
Markus Aleksy, University of Mannheim, Germany
Colin Atkinson, University of Mannheim, Germany
Barrett Bryant, University of Alabama at Birmingham, USA
Kuo-Ming Chao, Coventry University, UK
Remco Dijkman, University of Eindhoven, The Netherlands
Cléver Ricardo Guareis de Farias, University of São Paulo, Brazil
Ivan Ivanov, SUNY Empire State College, USA
Dimitris Karagiannis, University of Vienna, Austria
Dick Quartel, University of Twente, The Netherlands
Shazia Sadiq, University of Queensland, Australia
Boris Shishkov, University of Twente, The Netherlands
Ken Turner, University of Stirling, Scotland

Additional Reviewers

Enis Afgan, University of Alabama at Birmingham, USA
Fei Cao, Microsoft, USA
Hans-Georg Fill, University of Vienna, Austria
Peter Höfferer, University of Vienna, Austria
Dat Cao Ma, University of Queensland, Australia
Martin Nemetz, University of Vienna, Austria
Larry Tan, University of Stirling, Scotland

Supporting Organizations and Projects

INSTICC - Institute for Systems and Technologies of Information, Control and Communication

CTIT - Centre for Telematics and Information Technology

Freeband A-MUSE project - Architectural Modeling Utility for Service Enabling in Freeband

Table of Contents

Invited Speakers

What Can Web Services Learn from Business Process Modeling?

Dimitris Karagiannis

University of Vienna
Faculty of Computer Science
Dept. of Knowledge and Business Engineering
Brünner Straße 72, A - 1210 Vienna, Austria
dimitiris.karagiannis@univie.ac.at

Abstract. The research on Web-Services transformed the initial simple technical communication framework by integrating the Web-Service framework (WRSF) in the Grid framework (OGSA) and by merging Semantic Web standards with the Web-Service framework (e.g., SWSF, WSMF, OWL-S) to a formally enriched conceptual framework that - although it is only an implementation approach for Service Oriented Architecture (SOA) – is sometimes used as a synonym for a service-driven approaches. This implies that the Web-Service technology has now to deal with context-awareness, a formal description and methods for design, implementation and integration.
The established Business Process Modeling approach (a) provides solutions for modeling context awareness of Web-Services by including process information such as user categories, process status and task depending context, (b) supports the formal description via meta models, and (c) enables a methodical approach to design service oriented systems.
Web-Services can therefore be improved if the Business Process Modeling is seen as a process-driven approach to design Web-Service oriented systems, and as an integration platform.

Brief Biography

Prof. Karagiannis studied Computer Science at the Technical University of Berlin. From 1987 until 1992 he was business unit manager for Business Information Systems at the Research Institute for Applied Knowledge Management (FAW) in Ulm, Germany. In 1993 he founded the Department of Knowledge Engineering at the Institute for Computer Science and Business Informatics at the University of Vienna, focusing on Knowledge Management, Business Intelligence and Meta-Modeling. Prof. Karagiannis has published many scientific research papers in the field of Databases, Expert Systems, Business Process Management, Workflow Systems and Knowledge Management. He is the author of two books concerned with Knowledge Databases and Knowledge Management. Part of his research is done in the context of national and EU-funded projects. He established the Business Process Management Approach, which has been successfully implemented in several service companies. He founded the European software and consulting company BOC ITC Ltd (http://www.boc-eu.com), which realized the development and implementation of the business management toolkit ADONIS.

PAPERS

Architectures

Designing a Generic and Evolvable Software Architecture for Service Oriented Computing

Herwig Mannaert, Kris Ven and Jan Verelst

University of Antwerp, Department of Management Information Systems
Prinsstraat 13, B-2000 Antwerp, Belgium
{herwig.mannaert,kris.ven,jan.verelst}@ua.ac.be

Abstract. Service Oriented Architecture (SOA) is becoming the new paradigm for developing enterprise systems. We consider SOA to be concerned with high-level design of software, which is commonly called *software architecture*. In this respect, SOA can be considered to be a new architectural style. This paper proposes an advanced software architecture for information systems. It was developed by systematically applying solid software engineering principles such as *loose coupling*, *interface stability* and *asynchronous communication* to contemporary n-tier architectures for information systems in Java Enterprise Edition. The resulting architecture is SOA-compliant, generic and demonstrates to a high extent architectural qualities such as evolvability.

1 Introduction

In the last few years, Service Oriented Architecture (SOA) has been proposed as a new paradigm for building enterprise systems. Basically, the idea behind SOA suggests that systems should be built of services operating in highly networked environments. Since these services are modular and exhibit loose coupling, SOA should lead to evolvable systems. SOA is most often implemented by using Web Service technology. However, several authors emphasize that services can be composed of object oriented code, or even legacy code [1–3].

Building a SOA-compliant enterprise information systems for a specific organization is, however, not straightforward. From a technical point of view, one of the challenges is that SOA requires highly sophisticated designs to ensure that not only current, but also future requirements can be met. This means that the cost and effort in developing a full-scale SOA for a given organization is substantial, and for many organizations maybe even prohibitive. On the other hand, there seems to be a degree of similarity between the enterprise systems of most organizations. An indication of this is for example that most systems are based on a standard software package, with mostly limited customizations. This suggests that it may be possible to build an architecture for real-world, large-scale enterprise systems, which implements SOA-principles and can be used by a wide range of organizations.

In this paper, we propose an advanced, generic software architecture that could be used for building enterprise information systems. Initially, the architecture was developed for application domains such as large-scale satellite-based content distribution, monitoring and control of remote power units, and communications monitoring systems, but a prototype has shown its potential for building enterprise information systems. It was built according to contemporary n-tier architectures for information systems in the Java Enterprise Edition (Java EE) framework. The software architecture was built by systematically and thoroughly applying solid software engineering principles such as *loose coupling*, *interface stability*, and *asynchronous communication*. The resulting architecture is suitable for large-scale systems, generic and demonstrates to a high extent architectural qualities such as evolvability. The architecture is also SOA-compliant since it supports many SOA-principles, including loose coupling, reusability and abstraction [4]. The software architecture is independent from the underlying implementation technology (e.g., web services), but has been fully implemented in Java EE and is in use in several organizations.

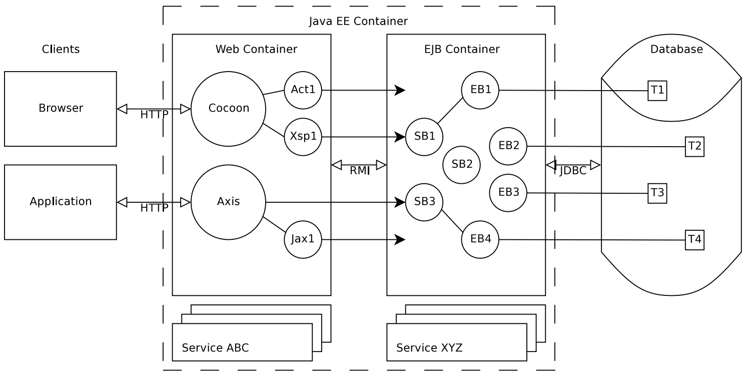
2 Software Architecture

SOA is a holistic concept spanning many research areas, from technical issues such as web services to management issues concerning business processes. However, our point of view is that SOA concerns essentially high-level design of software. This level is commonly called *software architecture*, and is a growing field of research within the area of software engineering [5]. More specifically, SOA can be seen as a new architectural style [6]. For example, Lublinsky considers SOA as an architectural style "*[...] promoting the concept of business-aligned enterprise services as the fundamental unit of designing, building, and composing enterprise business solutions. Multiple patterns, defining design, implementations, and deployment of the SOA solutions, complete this style.*" [7]. SOA attempts to increase modularity, thereby improving evolvability of the entire system. Also, considering SOA as a high-level design issue implies that SOA is more general than an implementation technology such as web services (i.e., web services is only one possible implementation technology for SOA).

Currently, client-server architecture [6, 8] is frequently used for developing information systems. Java EE, for example, is based on an n-tier client-server architecture. The software architecture we propose, is an attempt to SOA-enable these n-tier client-server architectures, or in other words, to extend them according to SOA-principles.

In order to visualize a software architecture, different views are necessary [9]. Each view differs in its intended stakeholders, and the system properties that are described. The *physical topology view* of the architecture that we propose is depicted in Fig. 1. Consistent with contemporary design principles, the concept of layering is adopted here. Each layer is highly cohesive, and loosely coupled to the other two layers. This principle ensures that modifications to a specific layer have no—or limited—impact on the rest of the system. This requires that each layer has an interface that shields the internal implementation details of that layer. This interface should remain stable in time (see Sect. 3.2). By introducing layers into the systems architecture, volatility can be better managed, as coding changes will not propagate across different layers.

Nowadays, information systems are generally composed of minimum 3 tiers: the *user interface tier*, the *business tier* and the *database tier*. In contrast to the traditional 3-tier design, we distinguish between 4 different tiers: the *client tier*, the *web tier* (con-



Legend: EB: Entity Bean; SB: Session Bean; Jax: JAX-RPC; Act: Cocoon Action Object; Xsp: XML Server Pages; T: Table

Fig. 1. 4-Tier Application Architecture.

taining for example Cocoon and Axis), the *EJB tier* and the *database tier*. Both the web and EJB tier are grouped in the Java EE container.

3 Guiding Principles

The architecture is based on several solid software engineering principles such as *loose coupling*, *interface stability*, and *asynchronous communication*. These principles are already known for quite some time. However, our main contribution consists of applying these principles in a systematic and thorough way. This allowed us to improve upon several architectural qualities such as evolvability, performance, security and availability [10]. In this paper, our focus will be on the evolvability of the architecture.

3.1 Loose Coupling

Loose coupling is an important principle in software engineering that aims to minimize the degree of interconnections (or coupling) between modules. If a module has a large number of connections to other modules, the module is also dependent on these other modules. As a result, the complexity of the system increases. We have applied the principle of loose coupling consistently throughout our architecture, by minimizing the number of interconnections between modules. In fact, we strive towards linking only two modules at the same time, in order to keep the complexity of the system to a strict minimum. We will provide several illustrations of this in the following sections.

3.2 Interface Stability

Evolvability is essential for an information system in order to accommodate changing requirements. In large-scale distributed systems, updating client applications following the release of a new version of a service provider is not always feasible. This new version could incorporate additional features and/or additional interfaces that are accessible to clients. In this section, we only consider *extensions* in the interface (i.e., the addition of parameters). This should however not affect the ability of existing clients—that will not use this new functionality—to keep accessing the service provider. We refer to this principle as *version transparency*. Hence, it is necessary that the interface of the service provider remains stable in time. We distinguish between two types of interface stability.

A first type is *strict-sense interface stability*. This type of stability requires loose coupling between modules that is completely implementation technology independent (i.e., it does not require that the service provider is based on a specific implementation technology such as Java EE). Consequently, the use of XML (e.g., web services that communicate via SOAP) is mandatory for the exchange of information between modules. This type of loose coupling is situated at run-time level, as recompilation of client applications is not required when a new version of a service provider is released. This type of loose coupling is preferable when there is a large number of distributed clients, or when the service client is located in a different unit of compilation than the service provider.

A second type of interface stability is *wide-sense interface stability*. This type respects the principle of loose coupling by only passing serializable objects with default constructors that only provide access to member fields through get and set methods. However, it allows imposing the use of a specific implementation technology (e.g., Java RMI). This type of coupling is situated at compile-time, since it requires recompilation of client applications upon the release of a new version of a service provider. However, coding changes to the service provider do not propagate beyond the service interface, i.e., modifications to a service provider should not require any coding changes to existing clients. Wide-sense interface stability can be a valid option when the number of clients is limited, or when clients are contained within the same unit of compilation as the service provider.

3.3 Asynchronous Communication

In general, service invocations tend to be synchronous: the client requests an operation from a service provider, waits for the provider to complete its operation, and receives the result of this operation. This is for example how web services essentially work. Synchronous communication however has some serious drawbacks.

First of all, the use of synchronous communication creates *temporal* coupling between modules [7]. This means that the client is blocked from the time that it issues the call until it receives a reply from the service provider. This may have negative performance consequences. It also requires that the service provider is available at the time the client issues the service request (i.e., the provider system is up and running, and there is network connectivity between service client and provider). Second, synchronous communication does not allow for the state of the transaction to be known. It is for example not straightforward to determine whether a service request has been submitted, but has not arrived yet at the service provider. Finally, when using synchronous communication, the client must incorporate additional knowledge about the underlying layers in the information system. This once again increases coupling, and as a result, the complexity of the system increases. For example, if a client has a user interface that retrieves data from a service provider, the user interface has to respond to the possibility that no network connection could be established to the service provider. However, it must also be able to react upon other errors that occur on the provider side, e.g., the fact that the database is currently down.

4 Architectural Patterns

We argue that additional structure is required, on top of standard component models and frameworks such as Java EE, Cocoon and Axis (see Fig. 1), in order to support the principles that were discussed in Sect. 3. Therefore, we have developed four different *architectural patterns* that are based on elementary object types, namely: *data objects*, *flow objects*, *action objects* and *connector objects*. Each of these patterns is cross-layer, since each pattern defines a number of objects located in several layers in Fig. 1. Although we do not claim that these patterns are the best possible solution, we have found them to be suitable for describing changes in a quantitative way [11], as well as automatic code generation [12]. It is important to note that these patterns are not solutions which are tied to a specific implementation platform. Instead, they are based on fundamental software engineering principles and concepts. In this paper, we illustrate how these underlying principles and concepts can be implemented in a certain technology (e.g., Java EE).

A code base has been developed within the Java EE framework. JOnAS is used as application server, while the Cocoon XML publishing framework provides the user interface. The code base consists of about 1200 Java classes, containing about 120 EJBs divided over 8 separate software components, and provides 5 different applications. These applications are divided in three different application domains: satellite-based content distribution [13], monitoring and control of remote power units [14], and communications monitoring systems (i.e., nurse call systems in hospitals and digital processing in a broadcast studio). Three components are shared by all five applications, the other components are currently confined to a single application. Given the genericity of this architecture (which is based upon the four architectural patterns), we are convinced that the architecture can be used to build enterprise information systems.

In order to build applications within the architecture, the universe of discourse (i.e., the relevant part of the real world) is modeled in terms of these four architectural patterns. For example, to develop an application for an on-line book store, data objects can be used to contain information on the books in the catalogue, connector objects can be used to generate sales reports and to provide the user interface, flow objects can be used to handle a sale, and action objects can be used to register payment through a credit card. We will now describe each of these patterns in more detail.

4.1 Data Objects

Data objects represent persistent objects in the real world that are stored in a relational database. Examples of data objects are *customer* and *order*.

Within the Java EE framework, data objects are implemented by using entity beans. For each object `<Obj>` that is stored persistently in the database, the Java EE framework requires the implementation class (`<Obj>Bean`), the interfaces for the lifecycle operations (find, create, and delete) (`<Obj>HomeLocal` and `<Obj>HomeRemote`), and the interfaces for the business methods (`<Obj>Local` and `<Obj>Remote`).

In addition to these five standard classes and interfaces, we include two additional *transport objects* for each persistent object. Transport objects are serializable objects that encapsulate data fields of corresponding data objects and only provide getters and setters to access each field. They also have a default constructor (without parameters), in which default values are set for all its member fields. A first transport object (`<Obj>Details`) contains all data fields of an entity. A second transport object (`<Obj>Info`) contains a subset of these data fields. The idea here is to include only those fields in the info-object that will be shown in for example listings and tables that display summary information.

These transport objects are essential to the architecture, since they support the principles of interface stability and version transparency. As a rule, only transport objects are allowed as parameters or return value in the interface of service providers. This allows for loose coupling—and version transparency—between service client and service provider. Our architecture supports both *wide-sense* and *strict-sense* interface stability. Wide-sense interface stability is implemented by exchanging serialized transport objects with remote session beans by using Java RMI calls. This design ensures that recompiling the client is sufficient when the service provider is extended in functionality through the addition of parameters (i.e., no coding changes to existing clients are required). Strict-sense interface stability is obtained by serializing the transport objects to XML format, and invoking the web service that corresponds to the session bean at the service provider[1].

Moreover, our architecture supports dynamically changing (i.e., at run-time) how clients will call a service provider interface. The client will either call the session bean over Java RMI, or the corresponding web service by using XML messages. How the client must invoke the remote service is stored in the database at the service provider side. This setting can be changed at run-time. This means that—theoretically—the degree of coupling between client and provider can be changed as well. However, given the fact that the client must be able to support Java RMI in this situation, it means that the client must be recompiled when the service provider is modified (unless the client will only use web service calls in the future). This means that such clients are not strict-sense version transparent. This feature however provides opportunities for future evolvability of the system, and to make decisions on the architectural qualities at run-time. For example, if one initially wants to maximize performance, service invocations can take place over Java RMI. If, at a later time, the network configuration changes and a firewall is placed between the client and the service provider, the client can be reconfigured to invoke the corresponding web service.

[1] Within Java EE, session beans can be made available as web services.

4.2 Connector Objects

A connector object is used to import and export data objects from and to the outside world. Connectors can be used to transform data objects from and to: the *user interface* (e.g., HTML), *files* (e.g., PDF), and *network protocols* (e.g., UDP, HTTP, SNMP). Connector objects for example generate an entry form for an entity in the database, or generate a report in PDF format.

Within the Java EE framework, a session bean (`<Obj>ConnectorBean`) is created for each data object that will be imported or exported. This bean depends on at least one implementation class for a specific protocol (`<Obj><Protocol>`). This class is not an EJB and is independent from Java EE. The `<Protocol>` is a variable that allows for alternative implementations for a specific connector (e.g., to provide multiple implementations for sending and receiving network packets over TCP or UDP). This supports the dynamic configuration of different protocols or formats through dynamic class loading.

This design further builds on loose coupling. By dividing the responsibility of the import/export functionality between the session bean and the implementation class, the complexity of the system is kept to a minimum. The connector object (i.e., session bean) is part of the EJB framework, and has knowledge about the data model of the corresponding data object. It has however no knowledge about the specific implementation of the external format or protocol. The latter responsibility is assigned to the implementation class, which however has no knowledge about the EJB framework. The same principle is used at the user interface. The Cocoon action classes for example have knowledge of Cocoon and the data model, but not about the underlying EJB container. As such, each object in the system only has knowledge about (is coupled with) maximum two other objects, hence minimizing complexity.

4.3 Flow Objects

Flow objects represent business processes, i.e., a sequence of steps in a workflow. Examples of flow objects are objects that handle the processing of a new order, or the registration of a new customer. In our architecture, a workflow is considered to be a sequence of actions (implemented by action objects, see Sect. 4.4).

Within Java EE, a flow object is an entity bean (`<Flow>OrderBean`) that stores the consecutive transitions required to execute a workflow. This bean also captures the current state of the workflow. Each transition is stored as a persistent object by using another entity bean (`TransitionBean`). This entity bean contains information such as the input and output state, and a reference to the session bean (i.e., action object) that implements the transition. The editing of the workflow is supported by the entity bean (`<Flow>OrderBean`) which provides CRUD (create, read, update, delete) functionality.

An important advantage of storing workflow persistently in a relational database, is that it allows for dynamic reconfiguration. Within our architecture, it is possible to update the workflow within the application through a web-based interface. Although the Business Process Execution Language (BPEL) is often used to describe workflow, the disadvantage of BPEL is that it doesn't directly support persistency, nor concurrent access with transactional integrity. More particularly, editing a BPEL file in XML format through a web-based interface is not trivial. However, in order to support BPEL specifications, it is possible to develop connector objects to import a BPEL file, parse the XML, and store its contents in a relational database.

4.4 Action Objects

Action objects are atomic steps in a workflow. Action objects perform operations on data objects, or external resources such as files. Examples of actions are encrypting a file, and performing a credit card validation.

In Java EE, action objects are implemented by using session beans. Similar to data objects, action objects require an implementation class (`<Act>Bean`), the lifecycle interfaces (`<Act>HomeLocal` and `<Act>HomeRemote`) and the business method interfaces (`<Act>Local` and `<Act>Remote`).

Similar to connector objects, we applied the loose coupling principle. This means that the action itself is implemented in a separate class (`<Act><Implementation>`), which has no knowledge of the Java EE framework. In order to increase flexibility and ensure loose coupling, `<Implementation>` is a variable that allows for providing several alternative implementations for a specific action (e.g., to support payments via various credit card companies using different interfaces). Since `<Implementation>` is a variable, it allows to dynamically choose between various implementations at run-time.

Some actions need to be performed regularly (e.g., every hour). For such actions, an EJB session bean (`<Act>EngineBean`) and an EJB entity bean (`<Act>ServiceEngineBean`) are created. The latter represents a persistent object that controls the time interval at which the action needs to be run, and also allows to start and stop the action. If the target of the operation is a persistent object that is represented by an entity bean (i.e., a data object `<Obj>Bean`), the state of the action can also be stored persistently as an entity bean (`<Obj>TaskState`).

The implementation of workflow through flow and action objects is fully based on asynchronous communication. This ensures loose coupling between service client and provider. As a result, different action objects implementing consecutive steps in a workflow do not communicate directly with each other. Instead, the input for a given action is stored in a database table. Each action has an agent that regularly polls the database table for incoming requests. When an outstanding request is found, the corresponding action is performed on the data. The output of this action is written to a second table in the relational database. The client (i.e., flow object) that has requested the action will also regularly poll the database for the result of the action. Once the result is available, it will be retrieved. This output can be passed as input to the next step in the workflow. It is clear that this design functionally and temporally decouples consecutive steps in a workflow. Another advantage of this design is that all actions and intermediate results of actions are logged in the database, and can be retrieved at any time. This allows for the creation of test data based on real operations that were performed by the system in the past, rather than artificially created data. This historic information may also be used for audit purposes.

5 Conclusion

In this paper, we have presented an advanced software architecture for information systems. The architecture is consistent with contemporary n-tier architectures, and demonstrates to a high extent several architectural qualities. The architecture has several important contributions.

First, the software architecture is generic, which is supported by several properties. For example, the application framework developed within this software architecture provides five different applications in distinct application domains. The architecture is also independent on the implementation technology (we have chosen to implement the architecture in Java EE). Additionally, it is possible to dynamically reconfigure several properties of the system at run-time, by using CRUD operations on the system itself. Examples are the degree of coupling between modules, and the workflows contained in the system.

Second, we have developed four different architectural patterns that are used as elementary building blocks within the architecture. This implies that the patterns can also be used for implementing meta-activities that represent common operations on services (such as discovery, selection and monitoring). These patterns are independent from a specific implementation platform, and are based on several solid software engineering principles such as loose coupling, interface stability, and asynchronous communication. By thoroughly and systematically applying each of these principles, we considerably increased several quality factors such as evolvability. This is illustrated by various characteristics of the architectural patterns. Transport objects (part of the data object pattern) support the notion of interface stability and version transparancy. This allows to extend their interface without requiring a recompilation of existing clients. Moreover, the architecture allows to choose at run-time between service invocations over Java RMI or web services, allowing for example to cope with changing requirements in the network infrastructure. Both the connector and action objects support the concept of dynamic class loading. This allows to provide additional implementations where clients can choose from. These patterns also support loose coupling and asynchronous communication, thereby separating the implementation as much as possible from the rest of the platform. The flow objects support run-time modifications to the workflow that is stored persistently in a relational database. This allows to update the workflow without any recompilation.

Third, the architecture is SOA-compliant, in the sense that it implements the aforementioned software engineering principles which also constitute the core of SOA, irrespective of the underlying implementation technology (e.g., web services).

Our goal is to further validate and extend this architecture in several ways. First, although we have implemented and tested the architecture in a number of settings, we plan to develop additional applications in other application domains. More specifically, we are convinced that this architecture is appropriate for building enterprise information systems, and will build on the current prototype to demonstrate this in more detail. Second, research can be performed on how the real world can be mapped to the four architectural patterns. Finally, we aim to identify additional patterns across the four architectural patterns that allow for the automatic generation of fully working code, called *pattern expansion*. A first pattern that was successfully expanded is the CRUDS pattern, which involves the generation of classes that implement data and connector objects, and is described in previous work [12]. In order to develop information systems in this architecture, the developer needs to define the actions and the data model of the application. Based on these elements, a considerable portion of the source code can be automatically generated through pattern expansion.

References

1. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of service-oriented analysis and design (2004) IBM Developerworks, on-line available at `http://www-106.ibm.com/developerworks/library/ws-soad1/`.
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: Service-oriented computing: A research roadmap. In Cubera, F., Krämer, B.J., Papazoglou, M.P., eds.: Service Oriented Computing (SOC). Number 05462 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
3. Marks, E.A., Bell, M.: Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology. John Wiley and Sons, Inc., Hoboken, NJ, USA (2006)
4. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
5. Shaw, M., Clements, P.: The golden age of software architecture. IEEE Software **23** (2006) 31–39
6. Shaw, M., Garlan, D.: Software Architecture—Perspectives on an Emerging Discipline. Prentice Hall, Upper Saddle River, NJ, USA (1996)
7. Lublinsky, B.: Defining SOA as an architectural style (2007) on-line available at: `http://www-128.ibm.com/developerworks/library/ar-soastyle/index.html`.
8. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading, MA, USA (1998)
9. Kruchten, P.: The 4+1 view model of architecture. IEEE Software **12** (1995) 42–50
10. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J.: The architecture tradeoff analysis method. In: Proceedings of the Fourth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98). (1998)
11. Mannaert, H., Verelst, J., Ven, K.: Towards rules and laws for software factories and evolvability: A case-driven approach. In: Proceedings of the International Conference on Software Engineering Advances (ICSEA'06), Tahiti, French Polynesia, October 29–November 3. (2006)
12. Mannaert, H., Verelst, J., Ven, K.: Exploring concepts for deterministic software engineering: Service interfaces, pattern expansion and stability. In: Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel, French Riviera, France, August 25–31. (2007)
13. Mannaert, H., De Gruyter, B., Adriaenssens, P.: Web portal for multicast delivery management. Internet Research **13** (2003) 94–99
14. Mannaert, H., Huysmans, P., Adriaenssens, P.: Connecting industrial controller to the internet through software composition in web application servers. In: International Conference on Internet and Web Based Applications and Services, Mauritius, May 13–19. (2007)

Architectural Models for Client Interaction on Service-Oriented Platforms

Luiz Olavo Bonino da Silva Santos, Luís Ferreira Pires and Marten van Sinderen

University of Twente, Architecture and Services of Network Applications Group
P.O. Box 217, 7500 AE Enschede, the Netherlands
{l.o.bonino, l.ferreirapires, m.j.vansinderen}@ewi.utwente.nl
http://asna.ewi.utwente.nl

Abstract. Service oriented platforms can provide different levels of functionality to the client applications as well as different interaction models. Depending on the platform's goals and the computing capacity of their expected clients the platform functionality can range from just an interface to support the discovery of services to a full set of intermediation facilities. Each of these options requires an architectural model to be followed in order to allow the support of the corresponding interaction pattern. This paper discusses architectural models for service-oriented platforms and how different choices of interaction models influence the design of such platforms. Service platforms' functionality provisioning can vary from a simple discovery mechanism to a complete set, including discovery, selection, composition and invocation. This paper also discusses two architectural design choices reflecting distinct types of functionality provisioning namely matchmaker and broker. The broker provides a more complete set of functionality to the clients, while the matchmaker leaves part of the functionality and responsibility to the client, demanding a client platform with more computational capabilities.

1 Introduction

Service-Oriented Architecture (SOA) is a paradigm for software architectures that fosters the creation of complex systems by using distributed pieces of functionality (services) accessible through a set of standards. In this architecture, services are provided to clients by services providers. Clients search for services by browsing a list of available service descriptions stored in registries. A service description contains information about a service, such as what the service does, how to access the interface, and which information should be supplied in order to use the service properly, among others. After discovering a suitable service, the client invokes the service interface in accordance with the information contained in the service description.

In this scenario, each client should be able to search in the available registries, decide which service best fits its needs and invoke the service using the published interfaces. To facilitate these tasks, service platforms can play an intermediation role between client applications and service providers. On the provider's side, a service platform can be beneficial by providing a mechanism for rapid creation, deployment and advertisement of services. Examples of facilitators for these activities can be found in

[9] and [10]. On the client's side, the platform can offer support for discovery, selection, composition and invocation of services, amongst others.

Focusing on the client's side, platform designers should choose the set of activities to be supported by the platform from the activities mentioned above. This choice influences the behavior of the platform and the interaction model between the platform and the clients. Therefore, an investigation of the possible platform behaviors concerning those interactions is necessary to identify platform requirements and architectural components. As a consequence, the platform roles in the interaction with the clients are identified.

In this paper we consider that the roles played by the service platform describe behavioral patterns followed by the platform regarding its interaction with client applications. Platform designers have at their disposal a variety of platform role's choices depending on what they intend the platform to support. A service platform acts similarly to the middle agents described in [5], in which the authors define several possible roles for the middle agents depending on how they solve the intermediation problem. In this paper we focus on two possible roles that illustrate opposite levels of functionality support: the *matchmaker*, which offers a simple discovery mechanism and, the *broker*, which offers more complete set of functionality, including discovery, selection, composition and invocation.

This paper is structured as follows: section 2 gives an overview of platform roles regarding client interaction, section 0 details the matchmaker role while section 4 details the broker role and section 5 concludes and points some future directions for this work.

2 Platform Roles

The architecture of service platforms can be defined in terms of the type of interaction to be offered to the client. Following this approach, the level of support to be offered and the choice of the platform role to be played determine the requirements for the service platform. Among the activities supported by service platforms we can include: discovery, selection, composition and invocation [1]. Here, we define as *level of support* the subset of aforementioned activities offered by the platform.

Among the different possible types of roles for a service platform we present in this paper two alternatives, namely *broker* and *matchmaker*. While the former offers a high level of support the later operates in a simpler way and leaves more responsibilities to the client application.

Fig. 1 shows the different interaction patterns applied in three service architectures. Architecture 1 is the basic architecture for Web Services where the service provider publishes the service descriptions in a registry, the client queries the registry for the descriptions of available services and, after selecting an available service, the client invokes the appropriate service. Architecture 2 places a matchmaker between the client and the registry. In this way the client sends the criteria of the desired service to the matchmaker, which searches the available registries for service descriptions matching these criteria. In the case of a positive match, the matchmaker returns the description of the discovered service to be invoked by the client. Architecture 3 presents an example of the broker role. In this example, the platform not only provides matchmaking facilities but also invokes the discovered services on behalf of the client. If necessary, the platform also performs transformations on the results received from the invoked services to comply with the data format requested by the client.

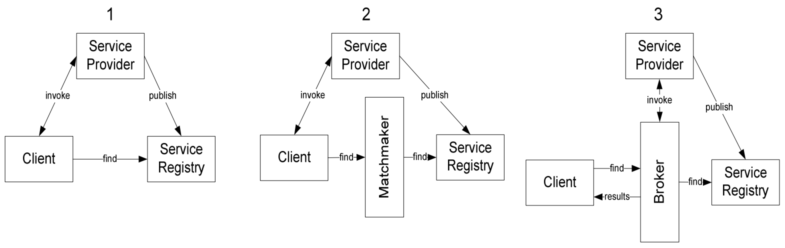


Fig. 1. Platform roles.

The matchmaker and the broker roles are discussed in more detail in the sequel.

3 Matchmaker

In the service matchmaking activity we have three distinct roles: a requester, a matchmaker and a provider [12]. The requester aims at finding services that offer the capabilities dictated to by criteria provided in terms of the desired service interfaces and properties [15]. The matchmaker has access to a set of services descriptions made available by providers and provides facilities to discover services based on the requester's criteria.

Early computational directories offer matchmaking facilities that provide mappings between names and addresses similarly to the white pages in telephone directories. Later on, a more advanced form of matchmaking emerged that supports search based on an entries' attributes allowing matches based on certain desired characteristics. This form of matchmaking resembles the yellow pages in telephone directories. One shortcoming of this approach is that the selection criteria are completely supplied by the requester, providing an asymmetric form of selection [13]. Work such as [14] suggests the introduction of symmetry in the selection process, in which the requester provides a description of the requested service and its capabilities as a client. The provider specifies its demand to the potential clients of its services. This allows the provider to select clients just as clients select services. This symmetrical matchmaking allows dynamic update of service descriptions at matchmaking time rather than at advertisement (publication) time.

A matchmaker acts like as if it were a provider of services of different providers. The requester does not have to interact with several providers or several service registries querying them for the descriptions of their services and then try to match the descriptions with the needed criteria. In a matchmaking environment, the requester sends the criteria to the matchmaker, which searches the services descriptions it has access to for a positive match. Having found services that comply with the given criteria, the matchmaker sends the service descriptions back to the client. The client then analyzes the services descriptions and selects suitable ones. After that, the client directly interacts with the services by invoking the service operations and receiving results.

Fig. 2 depicts a sequence of interactions between the client, the service provider and the matchmaker platform.

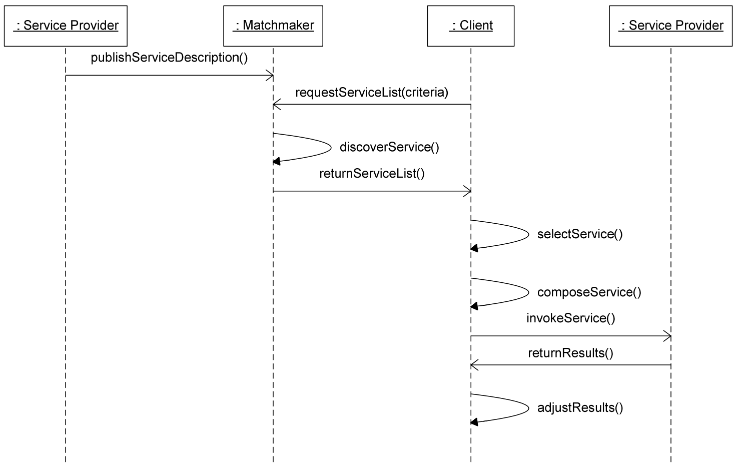


Fig. 2. Interactions in a matchmaking environment.

The level of support of the matchmaker indicates the architectural components of the platform. In case some functionality is not supported by the matchmaker, other applications or the client itself have to cover this functionality. Fig. 3 depicts a possible architecture for both the matchmaker and the client complying with the sequence diagram presented in Fig. 2. The matchmaker supports service publishing by the service provider through the *Service Publisher* component. The *Service Publisher* sends the received service description to the *Content Manager* component, which is responsible for storing it in an available registry (omitted in Fig. 3). The *Content Manager* shields the internal components from interactions with registries. The *Content Manager* can have access to several registries as well as acting as a client to other matchmakers.

A client requests the discovery of a service by calling the *Service Finder* component. The *Service Finder* requests a list of candidate services to the *Content Manager*. After receiving the list, the *Service Finder* tries to match the client's criteria against the candidate services to find the positive matches. If positive matches are found, the service descriptions are sent back to the client.

Since in this example the matchmaker does not support service composition, this task is expected to be performed by the client. Therefore, it is possible that the matches have been obtained by the partial satisfaction of the criteria, i.e., some of the properties given by the client have been satisfied but not completely by a unique service. In this case, the client needs to perform service composition by calling a *Service Composer* component. In Fig. 3 this component is internal to the client. The *Service Composer* tries to find a service composition that fully matches the criteria.

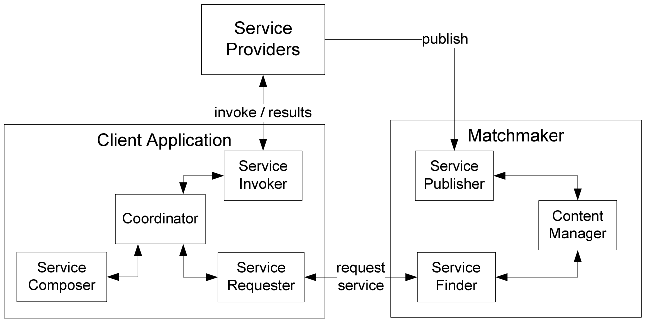


Fig. 3. Example of service composition performed by the client.

We consider now another example of matchmaker with a higher level of support than the one in Fig. 3. In this example, the service composition functionality is provided by the matchmaker. As it can be seen in Fig. 4, the *Service Composer* component has been moved from the client to the matchmaker but its functionality remains the same. The *Service Composer* still tries to compose the services that partially match the criteria into a service composition that fully matches the criteria. In case additional component services are required to complete the composition, the *Service Composer* needs to request the new services by providing other criteria. In the example where the service composition is performed by the client, the *Service Composer* requests the additional services to the *Coordinator* component. The *Coordinator* forwards the request to the *Service Requester* component which calls the *Service Finder* of the matchmaker with the new criteria. In the example shown in Fig. 4 where the composition is performed by the matchmaker, the criteria are passed by the *Service Composer* directly to the *Service Finder*.

The flexibility to assign functionality to the matchmaker or to the client shown in the example of service composition also holds for other functional components, such as the *Content Manager* or the *Service Finder*. Therefore, generic components can be developed to support these functions and the desired level of support for the service platform dictates where those components should be allocated.

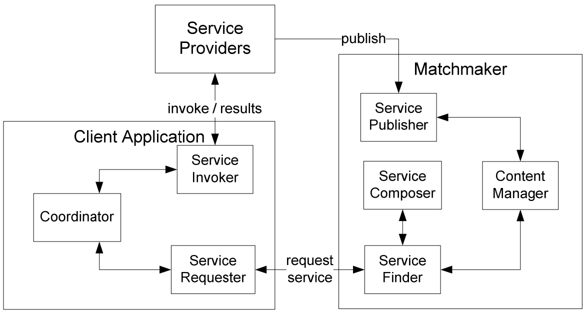


Fig. 4. Example of service composition performed by the matchmaker.

4 Broker

The following definitions of *broker* can be found:

- "An individual who gets buyers and sellers and helps in negotiating contracts for a client", Mortgage Magazine [2];
- "An agent who negotiates contracts of purchase and sale", Merriam Webster [3];
- "A person who buys and sells goods or assets for others", Oxford Dictionary [4].

Moreover, information brokerage can be loosely defined as a set of mediation capabilities and functions aiming to help sellers to broadcast or disseminate information about their products and services, and, at the same time, to assist the end users in order to better retrieve, select and compare the offered information about producers, products and services.

Mapping the definitions above to service-oriented computing we can define a broker service platform as *a platform that acts on behalf of a client application by discovering, selecting, composing and invocating services*. In this role, the platform offers a service selection mechanism, invokes the services on behalf of the client application, monitors the service execution and parses the results, possibly translating the output to client's required format. The broker can also perform service composition based on the client's service requirements and the service descriptions available to the platform.

Additional functionality can be assigned to the broker. For instance, if the service description contains semantic annotations, the platform should be able to perform a set of complex reasoning tasks [7], which includes interpreting service provider capabilities (service descriptions) and client applications' requirements. Moreover, the interpretation of the terms used in message exchanges can be performed by the platform.

Considering the broker role, an example usage scenario is the one in which the platform is available to clients that may request immediate provisioning of a service. The sequence diagram in Fig. 5 shows the interaction pattern between clients, service providers and the broker platform. Similarly to the matchmaker, the broker provides facilities for service publishing by the service providers. A critical difference between the matchmaker and the broker is that the latter acts as a surrogate of the client, i.e., the client requests the provisioning of a given service and the broker performs the necessary steps until the final result of the service, on the client's behalf. Even if the output request by the client is somewhat different from the output received by the broker after the invocation of the necessary services, the broker can perform a transformation to comply with the client's requirements. Examples of transformations are:

1. The client requests a service that returns the current temperature in Celsius for a given city. The broker finds a service that provides current temperatures of cities, but the output is in Fahrenheit instead of Celsius. In this case the broker can perform the output transformation by composing this service with another one that takes a temperature value in Fahrenheit and returns the value in Celsius;
2. The client requests a service that produces a given value in long number format. The broker finds the appropriate service but the output is in integer

number format. The broker can perform the transformation of the output by simply parsing the integer value into long and returning the transformed value to the client.

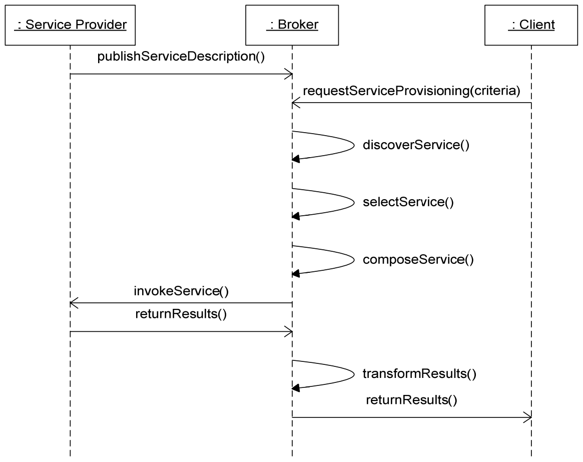


Fig. 5. Interaction pattern for the broker role.

Fig. 6 presents an example architecture of a broker. Here we see the components responsible for the functionality provided by the platform, namely the *Service Publisher* (service publishing), the *Content Manager* (service registry access and semantic repository access), the *Service Finder* (service discovery), the *Service Composer* (service composition) and the *Semantic Mediator* (semantic mediation).

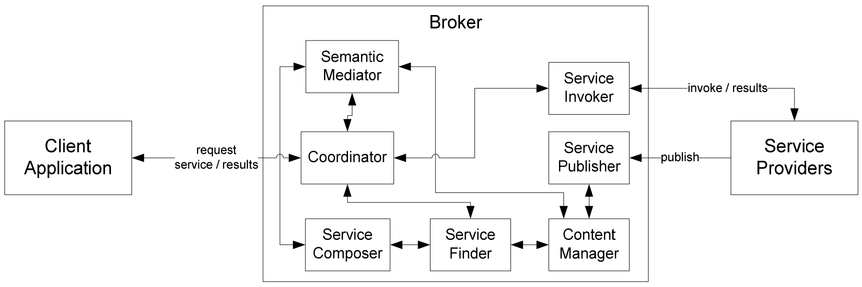


Fig. 6. Example architecture of a broker platform.

Additional characteristics of the broker platform role can be identified, such as:

- Fault tolerance and robustness: if a service becomes unavailable, the platform can try to find another suitable provider;
- Privacy, security and billing: since we assume that clients and service providers have agreed to trust the platform, the platform is the trusting central point for these entities. Therefore, clients do not have to directly interact

with services providers and vice-versa, and the platform can provide anonymization for both parties.

Nonetheless, being a centralized coordinator the platform can become a single point of failure as well. Techniques such as redundancy and clustering, among others, can be used to increase the platform's availability.

Unlike the matchmaker, the broker role has less room for exchanging functionality between the client and the platform. Although some auxiliary functionality, such as results transformations, could be placed on the client side, most of the functionality should remain on the platform side in order to preserve the surrogate characteristic of the broker.

5 Conclusions and Future Directions

In this paper we discuss the roles played by service platforms and the impact on the design of such platforms. To illustrate the discussion we present two choices of platform roles, namely the *matchmaker* and the *broker*. The main characteristics of each platform role are presented together with examples of architecture designs containing an overview of components providing the required functionality.

In this paper we identify the impact of the choice of platform role in the architectural design of the platform. In other words, the designer's choice of interaction pattern and the level of support of this platform implies in different assignments of architectural components to the clients and the platform. This paper addresses this impact for the matchmaker and broker platform roles. A platform design of the broker role for context-aware applications has been defined and proposed in [16].

Future directions of this work include the implementation of the suggested components and the instantiation of scenarios demonstrating the use of the presented platform roles. The exchange of functionality between client and the service platform, as suggested in the matchmaker examples, should be carried out and evaluated. Moreover, the requirements specification of both platforms (matchmaker and broker roles) should be detailed, in order to provide general guidelines for the design of such platforms.

Acknowledgements

The present work is co-funded by the Freeband Communication project A-Muse (http://a-muse.freeband.nl) and the Amigo Project. A-Muse is sponsored by the Dutch government under contract BSIK 03025. The Amigo project is funded by the European Commission as an integrated project (IP) in the Sixth Framework Programme under the contract number IST 004182.

References

1. Preist, C.: A Conceptual Architecture for Semantic Web Services. In Proceedings of the International Semantic Web Conference 2004 (ISWC 2004), pp. 395-409, November 2004.
2. Mortgage Magazine - http://www.mortgages-magazine.com/mortgage-glossary.htm.
3. Merriam Webster Dictionary – http://www.m-w.com.
4. Oxford Dictionary – http://www.askoxford.com.
5. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97), pp. 578-584, Nagoya, Japan, August 1997.
6. Chi Wong, H., Sycara, K.: A Taxonomy of Middle-Agents for the Internet. In Proceedings of the 4th International Conference on MultiAgent Systems (ICMAS 2000), pp. 465-466, Boston, MA, USA, July 2000.
7. Sycara, K., Paolucci, M., Soudry, J., Srinivasan, N.: Dynamic Discovery and Coordination of Agent-Based Semantic Web Services. IEEE Internet Computing, Vol. 8, n. 3, pp. 66-73, May/June 2004.
8. Piedad, F., Hawkings, M.: High Availability: Design, Techniques and Processes, Prentice Hall PTR, 1st Edition, December 2000.
9. Agarwal, V., et al. A Service Creation Environment Based on End to End Composition of Web Services. In Proceedings of the 14th International Conference on World Wide Web (WWW 2005), pp. 128-137, Chiba, Japan, 2005.
10. Srinivasan, N., Paolucci, M., Sycara, K., CODE: A Development Environment for OWL-S Web services. Technical Report CMU-RI-TR-05-48, Robotics Institute, Carnegie Mellon University, October, 2005.
11. Kawamura, T., et al, Web Services Lookup: A Matchmaker Experiment. IEEE IT Professional, vol. 7, n. 2, March/April 2005.
12. Decker, L., Williamson, M., Sycara, K., Matchmaking and Brokering. In Proceedings of the 2nd International Conference in Multi-Agent Systems (ICMAS'96), Kyoto, Japan, December 1996.
13. Facciorusso, C., et al, A Web Services Matchmaking Engine for Web Services. In Proceedings of the 4th International Conference on e-Commerce and Web Technologies, Prague, Czech Republic, September 2-5 2003.
14. Hoffner, Y., Facciorusso, C., Field, S., Schade, A., Distribution Issues in the Design and Implementation of a Virtual Market Place. Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 32, issue6, pp. 717-730, Elsevier North-Holland, New York, USA, May 2000.
15. Vausdevan, V., Augmenting OMG traders to handle service composition. Object Services and Consulting Inc., September 15 1998.
16. Bonino da Silva Santos, L.O., Semantic Services Support for Context-Aware Platforms, Master Dissertation, Universidade Federal do Espírito Santo, Vitória, Brazil, September 2004.

Concepts

Applying Component Concepts to Service Oriented Design: A Case Study

Balbir Barn and Samia Oussena

Thames Valley University, Computing Department
Wellington Street, Slough, SL1 1YG, UK

Abstract. This paper argues that appropriate modeling methods for service oriented development have not matured at the same pace as the technology because the conceptual underpinning that binds methods and technology has not been sufficiently articulated and developed. The paper describes an adaptation and enhancement of component based techniques to support the development of a service oriented method. As a result of the evaluation of using component concepts to support service oriented design, an integrated conceptual model describing how concepts from services and components are related is presented. The experimental data derives from a complex case study from the Higher Education Enterprise arena.

1 Background

Currently there is a focus on enterprise application integration using distributed architecture principles and in particular, there is a convergence to so-called Service Oriented Architecture (SOA) for application design and integration [19]. While enterprise systems have attained a degree of technical integration in many cases, the full benefits of business integration that could be gained from seamless support of business processes may only be partially realized. The developing principles around SOA are placing importance on a solid understanding of business processes and aligning developed or procured services to support those processes [11]. Thus methodologies that can support process led application development and assembly will acquire greater utility.

1.1 The Problem

Service Oriented Architecture is a disruptive technology because of the opportunity it provides to rethink the way systems are created and evolved. However there is still a relative lack of robustly applied and practical methodology support for such an approach. This observation has also been noted by Quartel et al [22] where they observe that technological developments should be supported by modeling methods and languages to support service-oriented design. One example where the methodology issues have been addressed to some extent is the recent work by Erl where there has been an effort to recognize that service-oriented analysis is an important element in the design of effective SOA [9]. However, here, the focus has been to derive services from a business process orchestration specification.

In contrast, Component Based Development (CBD) has reached a level of maturity where there is a significant body of knowledge addressing methodology requirements as well as technological issues. Given this, the research question addressed in this paper is:

"Can component based development concepts, methods and techniques support service oriented design?"

A service based architecture presents multiple concerns or architectural viewpoints. Consequently, the focus of the research question is further refined to address the functional viewpoint – that is, what an application (based on SOA) must do in order to support the business requirements of the user. Thus example areas that are not addressed in this paper are: the issues that arise at the boundaries between design and deployment of service-centric systems; and binding of services based on run-time monitoring of service-centric systems.

1.2 The Contribution of this Paper

This paper outlines an approach to service oriented design by drawing on the lessons learnt and the best practices from component based practices. The focus of this paper is on service identification and partitioning of applications into services and how such techniques can be captured in model based form. Further, the paper proposes an approach to business process partitioning that provides a model based migration strategy to process implementation using technologies such as Business Process Execution Language (BPEL). The paper also contributes an integrative view of services, components and business processes to further emphasize the benefits of pursuing this particular approach. Some of the observations and evaluation of software tools applied in the methods outlined also indicate that there are issues of tool usage which can help to inform tool selection and deployment.

The remainder of this paper is structured as follows: The reader is introduced to the background case study informing this research in section 2. Section 3 addresses the core of the paper and focuses on the comparison of concepts underpinning both component modeling and service oriented architecture with reference to relevant literature. Section 4 provides an evaluation of some of the results in the context of the case study and provides further details on the integrated conceptual model. Section 5 concludes the paper and outlines areas for further work.

2 Case Study

This section provides a short description of the context of the case study for which the business process modeling and subsequent application design was performed.

The e-Framework [16] is an initiative by the U.K's Joint Information Services Committee (JISC) and Australia's Department of Education, Science and Training (DEST) to build a common approach to Service Oriented Architectures for education and research. As part of this initiative, in 2005, JISC requested projects to develop reference models for a number of domain areas. The work described in this paper is derived from one of the projects.

The Course Validation (CV) process is one of the most important business processes within Higher Education Institutions (HEIs) and between HEIs and other institutions. New courses and the continuation of existing courses are the direct outputs of this process.

Further, the process is case-based, knowledge centric and highly collaborative. Each instance of the process is a case and will focus typically on different subject domains and therefore require different knowledge bases and experts to support the process. Only the essential framework (the rules and governance) of the process remain standardized.

The scope of the application domain is as follows. Course Validation can include the specification of new courses at various levels (e.g. undergraduate and postgraduate). Course Specifications address areas such as rationale, appropriateness, justification, marketing analysis, resources required, economic viability of the courses, and detailed descriptions of the courses in terms of outcomes, aims and objectives and so on. Much of the scope of course validation is determined by local institutional constraints (e.g. relationship to other courses and university regulations) but there are wider requirements that impose a significant overhead on the developmental process for validating new courses. These wider requirements are determined by the UK national bodies such as the Quality Assurance Agency (QAA) [21].

Even though HEIs may differ in the implementation of business processes to support course validation the constraints imposed by external bodies such as the QAA provide some standardization for the validation process and its outputs. These constraints are a basis for defining a canonical business process for supporting course validation.

A case study approach to the problem was adopted as there are several examples in IS research where there is evidence that case study based methodologies are well suited for exploring business processes in an organizational setting. Examples include those described in Huang et al [12] and Sedora et al [23]. Case studies provide an opportunity to take an interpretivist stance on how the systems and structures in place are based on the meanings of concepts and how people use those concepts. A case study also allows in-depth exploration of issues. However, given the nature of the course validation process it was important to get an understanding of how different types of institutions implemented their own course validation processes. Consequently we explored in depth, the course validation processes at four institutions.

After a period of business analysis, process models of course validation processes at each of the institutions were constructed. Accompanying information and data supporting these processes were also modeled. All modeling at this stage was performed using the IBM Rational XDE Toolset. The visual models were evaluated and an approach to synthesizing the models from each institution into a single canonical model was developed and then applied. This approach includes rules for identifying variances between processes and is described in more detail elsewhere [3].

The result was a pair of canonical models for the process and the information which were used as input to the software design and implementation stages to develop a set of software services that allowed us to automate part of the business process.

The remainder of the paper focuses on how the canonical process and information models were partitioned into a set of services to support the software design phase using a component based design approach.

3 Related Work and Approach

The approach taken in this research is based on three key principles: Firstly, the importance of systemizing the relationship between Component Based Development (CBD) and Services; secondly, to consider application partitioning from the perspectives of both business process partitioning and data partitioning; and finally the requirement to articulate a unifying conceptual model that addresses methods, business processes, components and services. Also critical to the approach taken in this paper, was the need to ensure that a model driven approach was followed. This was accomplished by, ensuring as far as possible, that all activities, artifacts and transformations were performed within one or more software tools. As the evaluation section indicates issues emerged during this process.

The central thesis of the paper argues that Component Based Development (CBD) provides a natural evolution to service oriented architectures because of the conceptual similarities and overlaps between the two software architecture approaches. In this section, the conceptual mappings between components and services are presented based on review of existing work. These mappings indicate the strong correlation between these two approaches thus indicating that it is instructive to look to CBD for appropriate methods and techniques to apply to SOA.

One important strategic distinction between the two approaches is the focus of integration strategies to address heterogeneous application architectures. While CBD at least, conceptually can be used to provide an application architecture that makes it possible to mix different implementation technologies, the evidence to date has indicated that this capability has not really been taken advantage of. For example, there are software component libraries for the Microsoft platform and similar libraries for the J2EE platform. SOA, on the other hand, has at its heart, standards and technologies that support interoperability. The use of XML based standards and protocols such as XML, SOAP, WSDL and UDDI allows services to be implemented in a particular technology while allowing access to the service from varying technical platforms using the so-called "wire" standards. A core common concept underpinning both CBD and service oriented design is the notion of an interface specification – a precise description of the behaviour of a software implementation. Interfaces can be used to provide wrappers to existing applications / modules such that it is possible to continue to use legacy applications in new technological environments. SOA is particularly suited to this approach and is potentially the most significant benefit arising from adopting a SOA strategy by an organization.

However, while a component can conceptually support more than one interface, a service has only one interface (in WSDL 1.1). Additionally, WSDL does not provide a mechanism for representing detailed behavioural semantics such as pre/post condition pairs. In separate work Estier et al demonstrate similar mappings and observations and in fact also use similar terminology such as "core". Their focus was on providing a "Contract" basis to service design rather than model based specification and generation [8]. Other work has developed a UML profile for describing WSDL (and therefore Web Services) to support WSDL generation from UML models – although the reported work proposes that implementation of add-ins for WSDL generation to commercial products such as Rational Rose is part of planned work [18]. Nonetheless this work while mapping to UML and not components would appear to further substantiate the validity of looking to CBD practice for methodology support.

The mappings indicate that we can leverage approaches and maturity of CBD practice to the design and implementation of web services by tailoring existing methods for software development.

Probably the most refined and detailed articulation of a component based method is work done by D'Souza and Wills in their description of Catalysis [7]. This method was later used to underpin the development of Cool:Spex [1, 2] (an early product to focus explicitly on application design using component based principles) and also other CBD methods [5]. Some of the CBD techniques from Catalysis and their derivatives that we can use include: component identification (or application partitioning), component specification, and component dependency management.

Application partitioning – the act of identifying discrete pieces of functionality into independent chunks of software is core to notions of component based development. However, CBD assumes a data centric view of partitioning. For example Erl describes such components as Entity Services. One proven approach to component partitioning and therefore service partitioning is that described by Cheesman and Daniels [5]. Their approach is based on using the information model (business concept model) as a starting position from which to make application partitioning decisions using an algorithm based on identifying core types and other types related to the core type.

However, despite the detail, complexity and scope available in the Catalysis method (and its variants), there has been relatively light attention to process modeling. This lesser emphasis is critical as process modeling is a crucial element for SOA where the orchestration of services to support an application is central to application assembly. The substantial standardization effort in business process execution (BPEL4WS) and prevalence of tools for choreographing applications from a set of services provides an indication of its importance.

This paper proposes that while data centric partitioning is one concern, it is also necessary to have a parallel and equivalent view of process partitioning. When processes are long, complex and require significant human intervention at various stages then the need for process decomposition is even more transparent. The case study used in this paper illustrates this point.

The Rational Unified Process (RUP) [14, 15] provides some guidance to this vis a vis the distinction business use cases standard use cases. This is an example of process decomposition. However, there is in-sufficient guidance and somewhat ambiguous rules for how business use cases map to use cases. Other ways of managing the modeling of process complexity for example to use roles – i.e. focusing only on the activities and their collaborations performed by specific roles [20] were considered inappropriate because the underlying process model was based on transformation (that is: input transformed by activity to output) rather than more communication/coordination views of processes.

One established approach is the use of "Event Consequences" – that is a business event is a trigger to a sequence of activities that are performed in response to the event. There is a rich body of knowledge which supports the notion of business process understanding using this approach for example [6, 24]. The set of activities that are triggered can then be viewed as a sub-process of the overall business process. Such a sub-process provides a better level of granularity for describing analysis scenarios for support the design and implementation stages of a software development process. An additional benefit of using events to partition a business process is the potential direct modeling transformation into BPEL specifications where there are modeling concepts for supporting events and their subsequent triggering of consequences of actions.

Summarizing, CBD practice provides a useful conceptual toolbox that need to be enhanced with a more detailed treatment of process modeling techniques in order to be useful to application development using SOA.

4 Case Study Example and Evaluation

Much of the anticipated benefits of a SOA approach are assumed to be the rapid assembly via orchestration of applications comprising one or more services. A pre-requisite for orchestration is a detailed understanding of the business process to be supported and the (ideally) model based specification. Thus the starting premise of the project approach is to precisely describe the business process using an appropriate modeling toolset.

The business analysis phase for the Course Validation (CV) domain produced two complex models – the process model and the information model.

4.1 Process Partitioning

Given the CV process complexity in particular, a way for decomposing the process into more manageable sub-processes was required. The CV process already had natural groupings of activities (these were distinct stages in the process) however, even these groupings were difficult to manage and it was necessary to define smaller sub-processes.

When a business process is a type of collaborative case process such as Course Validation then an especially useful form of partitioning is to identify situations in the business process where there is delay in the process because there is a need for an external event to occur. Once the event has arisen, new activities are undertaken. These groupings of activities are treated as sub-processes.

To support these sub-processes user scenarios were also developed. A user scenario is an evocative way of instantiating a route through a part of the business process. User scenarios are effective in extracting requirements because they express functionality in the language of users [4, 25] and for the implementation phase in this project they also provided additional context to development staff who were not involved in the original analysis stages of the project. During the development phase – the implementation team found the scenarios useful but still needed to elaborate additional sequence diagrams to further identify operation requirements.

Thus, in our process modeling we introduced the notion of sub-process scenarios. A sub-process scenario comprising one or more activities is triggered by an event such as a time or data event. This scenario and its accompanying user story can then be analyzed by the software designer to identify operations and allocate them to specific components/services.

4.2 Service Identification

Identification and modeling of services is the core of what is presented in this paper. It is argued that there is lack of methods and techniques to support service identification and modeling and currently most effort is focused advice and guidance on programming issues. It would appear that modeling advice is largely derived from object oriented analysis. Here it is proposed that given the conceptual closeness between services and components, it is possible to utilize techniques from CBD. In essence, the domain model or information model in our example is partitioned into "components" by firstly identifying types which are deemed to be core – that is business types or objects which are essential to the organization and then traversing associations to other types that are detailing – that are providing additional details to the core type. This subsetting provides a natural component boundary. Each component identified is then allocated an interface type which will house the operations for the component. This model thus corresponds to the service as follows: Component Interface is equivalent to Service; Core Type and detailing type are equivalent to the Port Types with their associated Messages and subsequently the elements and their schema. This approach bears comparison with Estier et al. who have similarly used CBD principles in their work on service contracts.

During the service partitioning activity a number of rules / hints emerged – the use of which has the potential for better quality component / service models. For example if two core types are related by a mandatory association (at both ends) it is still better to treat the core types as housed in separate components.

Often, components have a cross-relational dependency manifested via an intersecting detailing type. When this pattern occurs, one relationship is often "specifying" and the other is a "usage". In such situations, the intersecting type should always become the detailing type in the component that owns the "specifying" relationship. We anticipate further useful rules and hints as we continue to refine the component models.

4.3 Process Led Application Assembly

Once the components / services have been identified in this manner, the sub-process scenarios and their accompanying textual narratives are used to map activities from the process to an operation on a service. Service responsibility is based primarily on determining the types (information) being manipulated in the process and then allocating the service behaviour to the component/service that owns that type. The results of applying this technique to each of the sub-process scenarios is a set of components/services with behaviour allocated to them. In theory, then each component/service model can then be used to generate the required WSDL specification to support the web service.

In practice, this is where the violation of core model driven development principles unfolded. As intimated earlier, our key goal was to take a model driven approach to specify a business process and the accompanying information model; partition the models into a set of services; and then assemble services together to implement the business process. In order to do this, we would use software tools to model and generate the required elements. During the specification and design work toolset issues meant that the domain modeling (business analysis) was done using IBM Rational XDE as it was clear that the preferred toolset IBM Rational Software Architect (RSA) was not yet mature enough with its support for UML 2.0. However, the RSA implementation environment was considered to be superior to XDE so when the business analysis modeling and the partitioning into components/services was completed, the XDE models were imported without loss of data into RSA. Further modeling refinements were undertaken within RSA.

Issues during the design modeling phase also made it clear that it would have been better to create separate RSA models of each component (within the same overall project) as it made generation of WSDL and other XML easier and less error prone.

On generation to WSDL services, it became apparent that while the model structure to represent a component/service was correct (in that, all the required information to generate a WSDL spec was present), WSDL generation was not possible and the only generation of data that was achieved was the XSD schemas for the data requirements of the services. This represented a significant drawback to our proposed approach and the team is currently investigating alternative methods of WSDL generation with RSA..

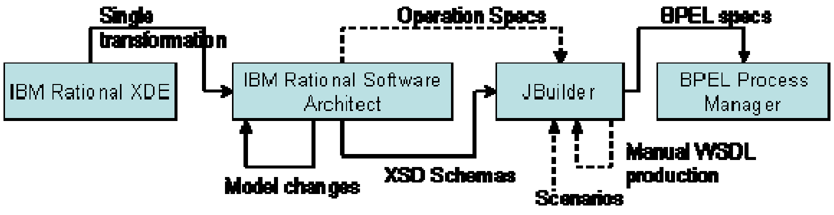


Fig. 1. Toolsets and transformations of models.

JBuilder was selected as the preferred toolset for designing the BPEL processes as again RSA and the Eclipse Plugin for BPEL process design did not fully support the design requirements. In this case the user interactions (user tasks) were not supported by the Eclispe Plugin. Within JBuilder, WSDL was handcrafted using further data from the text based user stories.

5 An Integrated Model for Component and Service Method Concepts

Having described the methodology for identifying and describing services from a business process basis, this section now proposes how the methodology and concepts need to be integrated to produce an overarching conceptual model which can be used to provide a basis for method refinement, tool construction and good practice.

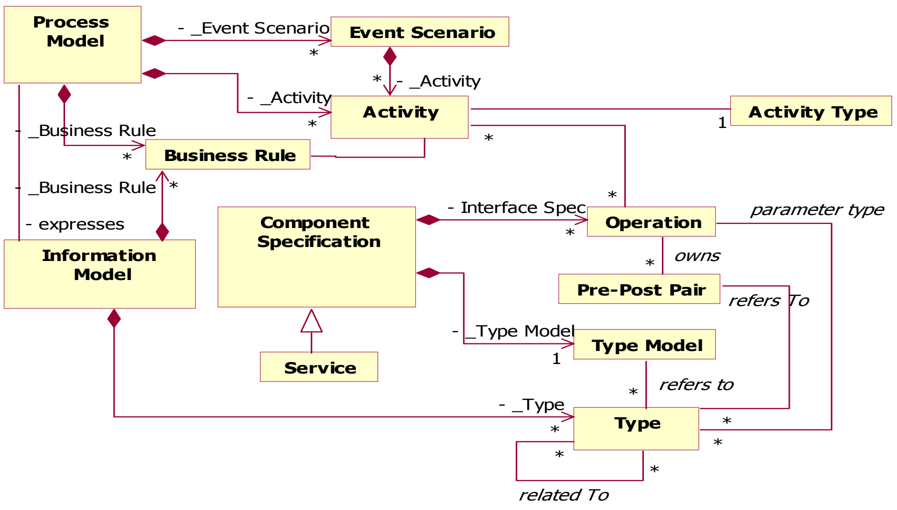


Fig. 2. SOA Integrated model.

The diagram above provides a UML model of the principal concepts involved. The Process model is decomposed into a set of Event Scenarios which are themselves a grouping of activities which have an ordering defined ultimately by UML semantics for activity modeling. An Event Scenario or Sub-Process provides a natural mapping to BPEL workflows.

In parallel, the domain information model is partitioned into a set of Services using CBD practice. A service has a set of operations which may or may not be specified by pre/post specification pairs. The types used by the operations of a service are grouped by the notion of an interface type model. These types can be used as the XSD schema for a WSDL specification. Activities are mapped to operations on the services via the Service/Activity matrix (or sequence diagram). Activities are classified as either manual (therefore not implemented, but their interfaces are specified), event receipt or normal. This classification is modeled by the Activity Type concept. This paper has not discussed business rules (constraints or invariants), in detail. In general, during the analysis phase, business rules were captured using Constraint annotations in the various models. However, we are planning further work to more formalize the capture of business rules using technologies such as XRules.

A further benefit of using an underlying meta model to describe methods is that development of techniques and concepts can be engineered allowing tailoring of method elements to support specific project requirements [13].

6 Conclusions and Further Work

This project set out to explore the interaction between SOA and model driven development. The case study and modeling approach has demonstrated that there is sufficient conceptual equivalence between component based approaches and service oriented architecture to warrant the use of CBD methods to identify and model services. From a complex process model, it was possible to partition the process into manageable sub-processes which could be orchestrated as BPEL workflows (albeit handcrafted).

The selection and deployment of the particular set of tools used in the project have been used to implement services and their process definitions with some success – with the primary problem centred around the model based generation to WSDL specs. The overheads and risks incurred by the use of such tools for bespoke application development using SOA remain significant and it is not clear how successful such a tool deployment strategy would be. It is possible that further investigation and increased expertise in tools such as RSA could help mitigate these risks.

Consequently, the project team is minded to conclude that SOA remains a significant challenge and perhaps best suited to application integration rather than bespoke development. As a result of this experiment, further work is being planned on the use of Business Process Management Toolsets such as Intalio Designer. One potential use of the conceptual model (after further research and validation) presented in figure 1 could be its use as a evaluation tool for the selection of tools (single or combined) However, SOA does require an emphasis on a business process modeling and research presented in this paper provides some enhancements to process modeling to ease the move from CBD to SOA. As we continue to develop services from new sub-process scenarios it is likely that we will refine our component partitioning strategy and the rules and hints to support the strategy. The use of the sub-process scenarios as model based input to Business process execution (BPEL) will also be the subject of further evaluation and study.

References

1. Barn, B.S., Brown, A.W., Cheesman, J.: Methods and Tools for Component Based Development. In Tools 98: Technology of Object-Oriented Languages and Systems, (1998)
2. Barn B.S., Brown A.W. Enterprise-Scale CBD: Building Complex Computer Systems from Components. In: 9th International Conference on Software Technology and Engineering Practice (STEP'99), Pittsburgh, Pennsylvania, USA (1999)
3. Barn, B.S., Dexter, H., Oussena, S., Petch, J. An Approach to Creating Reference Models for SOA from Multiple Processes In: IADIS Conference on Applied Computing, Spain (2006)
4. John Carroll. “Five Reasons for Scenario-Based Design” in Proceedings of the 32nd Hawaii International Conference on System Sciences – 1999.
5. Cheesman, J., Daniels, J. UML Components. Addison-Wesley (2001)
6. Cook, S., Daniels, J. Designing Object Systems: Object-oriented Modelling with Syntropy. Prentice Hall (1994)
7. D'Souza, D. F., Wills, A. C. Objects, Components, and Frameworks with UML: The Catalysis Approach. Object Technology Series. Addison Wesley, Reading Mass., (1999)
8. Estier, T., Michel, B., Reinhard, O. Modeling Services using Contracts: Identifying dependencies in Service-Oriented Architectures. In: EMMSAD 2006 Workshop – CAISE (2006).

9. Erl, T. Service Oriented Architecture – Concepts, Technology and Design. Prentice-Hall, USA (2005).
10. Frankel, D. Model Driven Architecture, OMG Press (2004)
11. Frankel, D.: Business Process Trends. BPTrends http://www.bptrends.com/ publicationfiles/07%2D05%20COL%20BP%20Platform%20%2D%20Frankel%2Epdf (2005)
12. Huang J.C., Newell S., Poulson B., Galliers R.D. Deriving Value from a Commodity Process: a Case Study of the Strategic Planning and Management of a Call Center. In: Proceedings of the Thirteenth European Conference on Information Systems (Bartmann D, Rajola F, Kallinikos J, Avison D, Winter R, Ein-Dor P, Becker J, Bodendorf F, Weinhardt C eds.), Regensburg, Germany. (2005)
13. Henderson-Sellers, B. Method engineering for OO systems development. Comm.. ACM 46, 10 (Oct. 2003), 73-78. DOI= http://doi.acm.org/10.1145/944217.94424
14. IBM-Rational: The Rational Unified Process (RUP),http://www-306.ibm.com/software/awdtools/rup/ (2001)
15. Kruchten, P. Rational Unified Process, Addison Wesley (1999)
16. E-learning Framework: http://www.elframework.org/ (2006)
17. Low, G. C., Rasmussen, G., Henderson-Sellers, B. Incorporation of distributed computing concerns into object-oriented methodologies; Journal of Object-Oriented Programming. (1996) Vol. 9, no. 3, pp. 12-20
18. Esperanza Marcos, Valeria de Castro, and Belén Vela (2003) "Representing Web Services with UML: A Case Study". In M.E. Orlowska et al. (Eds).: IC-SOC 2003, LNCS 2910, pp.17-27, 2003.
19. Ort, E. "Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools" http://java.sun.com/developer/technicalArticles/WebServices/soa2/ (2005)
20. Ould, M. A.: Business Process Management: A Rigorous Approach, BCS, ISBN: 1-902505-60-3 (2005)
21. QAA: http://www.qaa.ac.uk/
22. Dick Quartel, Remco Dijkman and Martin van Sinderen. "Methodology Support for Service-oriented Design with ISDL", ICSOC, 2004.
23. Sedera W., Rosemann M., Doebeli G. A process modelling success model: insights from a case study. In Proceedings of the Eleventh European Conference on Information Systems (Ciborra CU, Mercurio R, de Marco M, Martinez M, Carignani A eds.), Naples, Italy. (2003)
24. Texas Instruments. A guide to Information Engineering using the IEF™. TI Part Number: 2739756-0001. (1990)
25. Van Helvert, J & Fowler, C.J.H. (2004) Scenario-based User Needs Analysis. In Ian Alexander and Neil Maiden (eds) Scenarios & Use Cases Stories through the System Life Cycle. Wiley: London

An Approach to the Analysis and Evaluation of an Enterprise Service Ecosystem

Nicolas Repp, Stefan Schulte, Julian Eckert
Rainer Berbner and Ralf Steinmetz

Multimedia Communications Lab
Department of Computer Science
Technische Universität Darmstadt, Germany
repp@kom.tu-darmstadt.de

Abstract. Currently, the implementation of service-oriented concepts is one of the main activities of many IT and business departments throughout enterprises of various industries. Service-orientation as a concept is no novelty for many enterprises - many software systems and components offering technical and business functionality do comply with service-oriented principles. Nevertheless, the analysis, evaluation, and integration of existing services are often neglected in process models describing the implementation of service-oriented concepts. This paper describes an approach to the analysis and evaluation of those existing services to become part of the enterprise service ecosystem, which we call service inventory. The service inventory is realized as a generic extension to existing systems development methodologies, which allows its integration into the already used service-oriented methodology. The service inventory approach is based on Service-oriented Architecture research, principles from systems analysis and design, as well as on auditing principles.

1 Introduction

A current trend in software and enterprise engineering is the Service-oriented Architecture (SOA) paradigm, which can be used to design and develop complex IT systems. The core concept of SOA is the “service”, which can be understood as a self-describing encapsulation of domain-specific functionalities [1] [2]. Business processes and the applications supporting them can be built based on compositions of distributed and loosely coupled services.

Following the SOA trend many software vendors as well as IT service providers and consulting companies are jumping on the bandwagon, offering software, toolsets, and methodologies for the implementation of SOA in an enterprise and the system developments based on services. Especially in the area of middleware, supporting SOA (often referred to as the Enterprise Service Bus) and systems development methodologies for the implementation of SOA (the focus of this paper) there is a vast range of offers from almost all vendors. However, existing vendor specific systems development methodologies do not sufficiently consider the integration of an existing service ecosystem into the SOA, as they do not accept the fact that many principles and aspects of SOA are already in use. Furthermore, many software systems and components in use offer technical and business functionalities that already comply with service-oriented principles. Those existing service have to be found and integrated into the SOA.

In this paper, we describe an approach to the analysis and evaluation of an existing enterprise service ecosystem and its services, which we call “service inventory” according to the concept of the inventory process in financial accounting. Parts of the approach are the result of a research project in cooperation with an industry partner. The goal of this research was not to create “yet another” systems development methodology but to develop a generic enhancement of existing SOA systems development methodologies. Furthermore, the development of a tool for the analysis and evaluation of services (“auditing of services”) was also an objective. Intension behind the tool was to allow the creation of a service inventory by an experienced third party in a timely manner. Both the approach itself and the tool have to be customized with respect to data models and terminology in order to fulfill the needs of a particular enterprise.

The rest of this paper is structured as follows. In the next section, we will provide definitions important for the further understanding of the paper. The successive section introduces aspects of service-orientation, which are the foundation of the evaluation framework included in our service inventory approach. The service inventory approach is described in a separate section based on a generic process, which can be part of existing commercial systems development methodologies. The paper closes with a conclusion and an outlook on future work.

2 Basics and Definitions

2.1 Services and Business Processes

As already discussed in the introduction, we define services as self-contained encapsulations of domain-specific functionalities. The whole of services, which can be used by an enterprise, is referred to as “enterprise service ecosystem”, which includes both internal services and services obtained from external sources. The services can be composed to execution plans containing the functionality of the business process [3] [4] and subsequently be executed by a business process engine. The possibility of a mapping between business processes and services offering the needed functionalities are a precondition for our work. We will not further discuss this topic in this paper.

Every relationship between a service provider and a service requester, no matter if a service provider is internal or external, has to be documented in the form of a contract, a so-called Service Level Agreement (SLA), describing the most important aspects of the relationship. This contract can be explicit, i.e., described in a dedicated document, or implicit by simply using a given service. Especially in business critical processes the management and enforcement of SLAs is crucial [5]. SLAs can be part of the data, which is examined during the service inventory process.

2.2 Service Inventory

According to the common understanding of the term “inventory” in accounting as both “a detailed list of all the items in stock” and the “making (of) an itemized list ...” (following the definition of the term in WordNet), we also use the term “inventory” in this paper both for the process of taking stock as well as for the result of this process.

If applied to the concept of an enterprise service ecosystem, services (here: services provided by the enterprise in scope) also can be seen as intangible assets of an enterprise, which have to be valued. The stocktaking of existing services, their analysis, and evaluation is therefore called “service inventory”. Depending on the point in time and the frequency of a service inventory, we can distinguish between a periodic (i.e., annual or project initiated service inventory) and a perpetual service inventory (i.e., continuous tracking of services). We assign no monetary value to a service during the service inventory. Instead, the value of a service is measured based on its ability to be integrated in the SOA, which should be implemented by the enterprise. To be more precise, the measurement is founded on the assessment of several aspects of services, which are discussed in the next sections.

In addition to the definition of the service inventory process, a service inventory also describes the result of the taking stock process. Therefore, the service inventory also represents a listing of all services at a given point in time. It can be managed by a service repository. The description of services itself as well as the format of the listing of services is out of scope of this paper. We will focus on the service inventory as a process for the rest of the paper.

3 Service Aspects

This section about service aspects describes the foundation for the service inventory process. Based on the aspects given in Table 1, the ability of integration in a SOA is analyzed and evaluated. The service aspects were derived from different sources, i.e., the common principles of service-orientation presented by Erl [6], the rules for architectural design of enterprise IT by Voß et al. [7], the standardized specification of business components by Ackermann et al. [8], as well as from our experience in SOA projects.

Of further importance is the documentation of a service. It is the foundation for all the aspects mentioned above - without a sufficiently complete and comprehensible documentation the assessment of the services is not possible. To improve the usability of the service inventory process we created a criteria catalog, which represents the content and intension of the given service aspects. Every criterion is formulated as a question, which can be easily answered based on a given range of possible answers (mostly “yes”/“no”, where “no” signalizes a deviation from the targeted state).

4 Our Approach to the Analysis and Evaluation of Services

In this section, we describe the service inventory process. As a foundation for the service inventory process, we will also present a generic systems development process with respect to the implementation of SOA, in which the service inventory process can be integrated.

Table 1. Service aspects as the foundation for the service inventory.

Service aspect	Description
Reusability	Describes the ability of a service to be used without changes in different scenarios than the ones it was specified for. Only changes in the parameterization of the service are acceptable.
Granularity	Depicts the functional range as well as the complexity of a service. Services of coarse granularity offer functionalities of an entire business domain, e.g., the implementation of an entire business process, whereas services of fine granularity offer base functionalities useful in various scenarios.
Autonomy	Characterizes the independence of a service from other services as well as from other resources. Furthermore, autonomy also describes the uniqueness of a service with respect to its functionality, i.e., there are no two services in one enterprise service ecosystem with the same functionality.
Context independence	Describes the property of a service to operate without any context or state information. Every call/execution of the service provides the needed information to operate. There is no keeping of state information or session concept. Furthermore, a service has to offer compensating functionalities in order to be independent from external intervention in case of an exception.
Degree of coupling	Specifies an additional measurement for the independence of a service, aiming towards a preferably loose coupling of services, i.e., services can be exchanged on the fly without the consideration of any dependencies.
Information hiding	Neither information about technical details of the implementation nor information with business or security critical character should be visible to service requesters, especially if the service requesters are from outside the enterprise.
Discoverability	Characterizes the need to document a service and make it identifiable in order to be found. Additionally, discoverability is also crucial in order to avoid parallel developments of the same functionalities in different departments or the unnecessary purchase of services due to the lack of knowledge.

4.1 Prerequisites for the Approach

In order to apply the phases of the service inventory process to a given situation, we need a sufficiently complete set of data about the service in scope. In contrast to the inventory process known in accounting, we need a description of the subject we are looking at because of its intangible character.

Based on experience from projects in the finance and telecommunications sector, we observed that the following information about a service is usually available for an assessment:

- Non-formal description of the service functionality.
- Description of both service provider and service requester.
- Service Level Agreements describing the non-functional properties a service provider is willing to offer.

Furthermore, the following information is helpful, if available:

- Formal description of the service functionality, e.g., in form of a semantically enhanced interface description.
- Description of prospective service requesters and usage scenarios.
- Classification of the service based on a service model used by the enterprise.
- Documentation about the business processes, in which the services are/can be used.

It is not always possible to start the service inventory process with a complete set of the information described above. In case of missing or ambiguous information with respect to the service aspects, interviews or walkthroughs of the documentation in cooperation with members of staff should be used to gather the missing information.

4.2 A Generic Systems Development Approach for the Implementation of SOA

As we stated before, the service inventory process is not a standalone process. It has to be embedded in an existing systems development methodology for the implementation of SOA and respective process. Therefore, we will first present a generic systems development process in which the service inventory can be integrated. For this purpose, we analyzed different vendor specific systems development methodologies for the implementation of SOA, which we worked with before, i.e., methodologies, and processes of IBM, Software AG, and IDS Scheer. Based on the similarities of the models and the basic phases of the systems development processes discussed in Software Engineering and Systems Development literature (e.g., [9] and [10]), we derived a simple generic process for the implementation of SOA, taking full account of the given service ecosystem. Both the generic process and the service inventory process are modeled using the Business Process Modeling Notation (BPMN) [11], a standard for the description of business processes broadly used in the area of SOA, in order to simplify the integration with processes of different vendors or those already used by the enterprise.

The generic systems development process consists of five core phases, which all are sub-processes themselves (see Figure 1). For simplification, we do not model iterations of single phases or sub-processes as parts of the process in this paper, but they are possible and valid in our process. The analysis phase is depicted in-depth, as the service inventory process is part of it. The phases of the process are described in Table 2 in more detail.

4.3 Phases of the Service Inventory Process

As stated in the previous section, the service inventory process is part of the analysis phase in the generic systems development process for the implementation of SOA. We can distinguish four phases of the service inventory process, which are depicted in Figure 2. An in-depth description of the single phases is subject of Table 3. The result of the service inventory process can be directly used for the design of a SOA-based system, as it describes services, which can be reused. The four phases are intentionally kept generic in order to allow the adaptation to the current project context. Unfortunately, there are also no generally accepted recommendations for service aspects, e.g., for the granularity of a service, which could be used as a guide for the process.

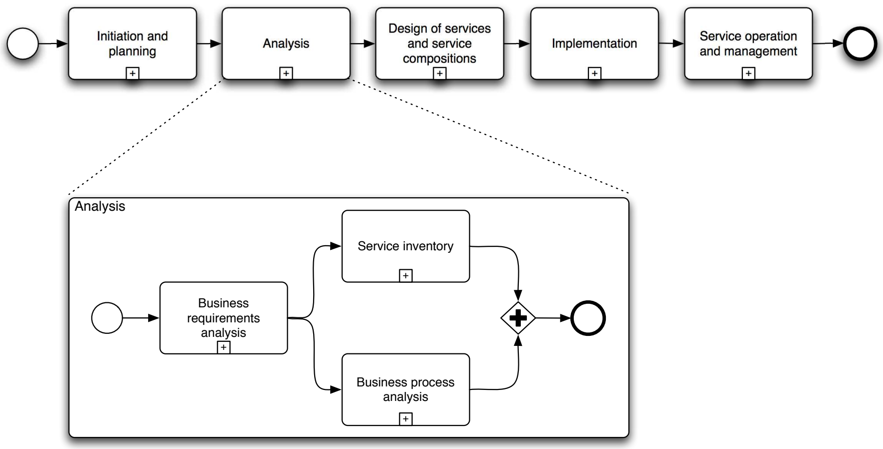

Initiation and planning
Analysis
Design of services and service compositions
Implementation
Service operation and management
Analysis
Service inventory
Business requirements analysis
Business process analysis

Fig. 1. Generic systems development process for the implementation of SOA.

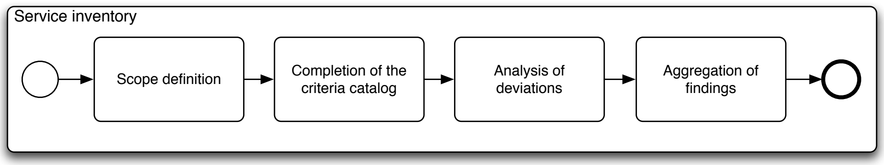


Fig. 2. Overview of the service inventory process.

Nevertheless, the service inventory process is not only an academic concept. In cooperation with a German IT consulting firm we developed an Excel based criteria catalog containing 28 questions, which can be mapped to the seven service aspects. We further integrated the service model of the firm, which classifies services into four categories. As a further enhancement of the generic process and criteria catalog, we created examples for the evaluation of the criteria to allow a quick access to the approach. Finally, we estimated the time needed for the completion of every single criterion in order to bundle the criteria into three preconfigured tests of different duration. Currently, the approach is used in first SOA projects.

4.4 Service Inventory Process by Means of an Example

In this section, we want to give an example of the service inventory process. In the first phase of our example, the objective of the service inventory is set to a check of service reusability. Due to this fact, the criteria of the catalog are the most important, i.e., receive the highest weights, which are related to the service aspect of reusability. A single critical deviation of a criterion related to reusability leads to an “orange” rating of the service, classifying it as only usable with restrictions. More than one critical deviation will lead to a “red” rating and therefore to the exclusion of the service from the

Table 2. Phases of the generic systems development process for the implementation of SOA.

Phase	Description
Initiation and planning	In this initial phase, problems with respect to the business and its processes are identified. The identification as well as the following sketching of a solution based on SOA is done on a high level, which is not sufficient for an implementation. If in this phase the decision is made to start a SOA-based project, further scoping and resource planning has to be carried out, e.g., staffing, budgeting, or the planning of milestones.
Analysis	Subsequent to the initiation and planning phase, the business needs and processing requirements as well as the as-is situation of both the business processes and the underlying enterprise architecture have to be examined in detail during the analysis phase. In this phase, a service inventory has to be performed to get an overview of services already in use, which could be integrated into the solution.
Design of services and service compositions	Based on the results of the analysis, we now can design services and compositions of services in order to implement the needed business processes. During this phase, not only new services have to be designed. Moreover, the focus of a service-oriented approach is on the reuse of existing services. Both the existing and new services are combined in the form of service compositions, i.e., parts lists forming blueprints of the business processes to be implemented.
Implementation	The blueprints and designs of business processes, service compositions, and services from the last phase are implemented in this phase. The implementation is realized on different levels of abstraction. Services are implemented for example using traditional programming languages and Web Service technology. Furthermore, service compositions also have to be implemented in a format, which is machine-readable and -interpretable. For this purpose, higher level composition languages are used, e.g., the Business Process Execution Language (BPEL) in the Web Service domain [12]. Service implementations and execution plans of business processes based on services, which can be processed by business process engines, are the outcome of this phase.
Service operation and management	As the last phase of the generic systems development process for the implementation of SOA, we describe the service operation and management phase. This phase contains aspects of both a systems development process and a service lifecycle model. In this phase, the set of services implementing business functionalities is executed. The monitoring and measurement of the execution often results in a need for improvement and optimization of the developed system, so that additional iterations of the systems development process have to be performed.

SOA. In the next phase, the adapted criteria catalog is completed based on a review of the given documentation. The following questions are examples of criteria with respect to the reusability of a service:

– *Does the documentation provide information about the data types and formats used to invoke the service?*

Table 3. Phases of the service inventory process.

Phase	Description
Scope definition	In this initial phase, the aims and objectives of the service inventory have to be defined depending on the current situation of the enterprise. Therefore, the scope of the service inventory has to be set, e.g., what services have to be assessed and what amount of time is available for the service inventory. Furthermore, the properties of a service, which are needed to fulfill the goals of the targeted project, have to be specified in terms of the seven service aspects. In addition, the aspects have to be weighted among each other with respect to their relevance for the project and deviations from the targeted state have to be defined. Finally, it has to be specified how many deviations of what severity lead to negative rating of the service for the project ("definition of materiality").
Completion of the criteria catalog	Based on the scope of the service inventory, in this phase the questions in the criteria catalog have to be completed during a review of the documentation, an interview, or a walkthrough. The criteria catalog is completed per single service.
Analysis of deviations	In this phase, the deviations detected in the previous phase have to be analyzed with respect to their severity. The deviations are classified into the three types "minimal", "moderate", and "critical", based on the definitions provided in the initial phase.
Aggregation of findings	In the final phase of the service inventory process, the results of the previous phases have to be aggregated for every single service. The result of this phase is a statement about the usefulness of the service for the project and its ability to be integrated in the SOA. There are three possible outcomes of the service inventory process for every single service in scope, which are classified into "green" ("service is useable without restriction"), "orange" ("service is useable, but with restrictions"), and "red" ("service is not useable" or "not a service").

– *Does the description of the service provide mapping information of the service functionality to the classification framework used by the enterprise?*

Subsequent to the completion of the criteria catalog, the deviations detected in the previous step have to be analyzed in detail during the third phase. In our example, the non-existence of the documentation of both data types and format for the service invocation is classified as critical. Finally, the singular ratings of the service have to be aggregated. One single critical rating results in an overall classification of the service as "usable, but with restrictions" ("orange"), because of the lack of adequate documentation for data types and formats is easily remediable.

The service inventory process is documented in a dedicated format, containing information about the assessed services, the scope of the service inventory, the auditor performing the service inventory, the findings as well as the time needed for the service inventory. Based on the documented amount of time needed for both the service inventory process and single criteria, the service inventory process can be customized in order to fit the needs with respect to the time restrictions of the project.

5 Conclusion and Outlook

In this paper, we presented an approach to the analysis and evaluation of an enterprise service ecosystem, which is called service inventory. The service inventory is not a new systems development methodology but an enhancement of existing approaches with a strong emphasis on the current situation of an enterprise planning to implement a SOA. Our approach details the phases of vendor specific processes coping with the analysis and integration of existing services. All of the vendor specific processes address existing services, but they do not address how to analyze and evaluate them with respect to their potential use for the SOA implementation project. Based on the generic character of our approach it is highly adaptable to the needs of an individual enterprise.

In the future, we have to specify and formalize the phases of both the generic process and the service inventory process in more detail, whereas the differentiation of the phases in both processes will not be changed. Additionally, the service aspects, which are the foundation of the service inventory process, will be improved further with respect to recommendations for the individual service aspects. For this, we are currently preparing a multi-participant case study to evaluate best practices for service granularity in different industries. Of further importance is also the specification of services itself. We will evaluate existing specification frameworks for the description of services in order to integrate such a framework into our approach. Additionally, we plan to integrate our criteria catalog as well as the process into a web-based application in order to support the service inventory process.

Acknowledgements

The work is supported in part by the E-Finance Lab e.V., Frankfurt am Main, Germany.

References

1. Krafzig, D., Banke, K., and Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, Upper Saddle River, USA, 2005.
2. Papazoglou, M. P.: Service-Oriented Computing: Concepts, Characteristics and Directions, 4th International Conference on Web Information Systems Engineering (WISE 2003), IEEE, Rome, Italy, 2003, pp. 3-12.
3. Repp, N., Berbner, R., Heckmann, O., and Steinmetz, R.: A Cross-Layer Approach to Performance Monitoring of Web Services, ECOWS 2006 Workshop on Emerging Web Services Technology, IEEE, Zurich, Switzerland, 2006: CEUR Workshop Proceedings, vol. 234, ISSN 1613-0073, http://ceur-ws.org/Vol-234/paper2.pdf.
4. Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E., and Steinmetz, R.: An approach for the Management of Service-oriented Architecture (SOA) based Application Systems, Enterprise Modeling and Information Systems Architectures (EMISA 2005), Klagenfurt, Austria, 2005, pp. 208-221.
5. Berbner, R., Heckmann, O., and Steinmetz, R.: An Architecture for a QoS driven composition of Web Service based Workflows, Networking and Electronic Commerce Research Conference (NAEC 2005), Riva Del Garda, Italy, 2005.

6. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, USA, 2005.
7. Voß, M., Hess, A., and Humm, B.: Towards a Framework for Large Scale Quality Architecture, 2nd International Conference on the Quality of Software Architectures (QoSA 2006), Västeras, Sweden, 2006: Internal Report 2006-10, Universität Karlsruhe, ISSN 1432-7864.
8. Ackermann, J., Brinkop, F., Conrad, S., Fettke, P., Frick, A., Glistau, E., Jaekel, H., Kotlar, O., Loos, P., Mrech, H., Ortner, E., Overhage, S., Raape, U., Sahm, S., Schmietendorf, A., Teschke, T., and Turowski, K.: Standardized Specification of Business Components, Gesellschaft für Informatik (German Informatics Society), Working group 5.10.3, Augsburg, Germany, 2002: http://www.fachkomponenten.de, accessed at 2007-04-22.
9. Whitten, J. L., Bentley, L. D., and Dittman, K.: Systems Analysis and Design Methods, 6/e, McGraw-Hill, Columbus, USA, 2004.
10. Singh, S., and Kotze, P.: An Overview of Systems Design and Development Methodologies with Regard to the Involvement of Users and other Stakeholders, Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT 2003), South Africa, 2003, pp. 37–47.
11. Object Management Group - Business Process Management Initiative: BPMN 1.0, Final Adopted Specification, February 6, 2006: http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf, accessed at 2007-06-07.
12. Organization for the Advancement of Structured Information Standards: Web Services Business Process Execution Language Version 2.0, Committee Draft, May 17, 2006: http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm, accessed at 2007-06-07.

Integrated Governance of IT Services for Value Oriented Organizations

Antonio Folgueras Marcos, Belén Ruiz Mezcua and Ángel García Crespo

Carlos III University, Escuela Politécnica Superior, Computing Department
Av. Universidad 30, Madrid, Spain, 28911 Leganés
afolguer@inf.uc3m.es, bruiz@inf.uc3m.es, acrespo@ia.uc3m.es

Abstract. This paper shows a latest generation model for the management and governance of the technologies and Information Systems (IT) in the organizations. IT governance is the key to achieve high level of maturity in the SOA Maturity Model. Currently there are standards and methodologies of international character that cover in detail the different critical aspects of the Information Technologies' governance such as CobiT, Itil, ISO20000 and Balanced Scorecard for IT. This governance model starts from the knowledge acquired in the mentioned standards and allows carrying out the tacit and strategic governance of all the activities in an information systems department. The model depicted in this paper includes: the monthly and tacit control of the IT processes, the system's portfolio management, the IT strategy planning and the alignment of the strategies with the operations (four alignments because considers: systems and business). This model is called IG4 (Information Governance Four Generation) due to the fact that it includes important improvements on the classic management and governance IT models.

1 Introduction

The objective of this paper is to define a standard to cover the IT Governance in all kind of service oriented organizations. With this standard the CIOs can plan, manage and control all the IT processes in an integrated way. Despite the Information Technologies' Governance (IT) being a relatively new area, it boasts interesting tools that provide best practices to cover five basic aspects:

1. Audit and control of the IT internal processes by means of metrics. The IG4 model reaches the monthly level, the tacit and strategic, because the daily control of the operations (for example the incidences) exceeds the IT governance tools' reach.
2. Better practices that provide a comparison with the IT work of the organizations in similar sectors and environments.
3. Double aspect alignment: business and technologies as well as strategy and operations.
4. Strategy planning and simulation.
5. Added value techniques of investment and decision analysis to manage the projects / applications to deal with.

This four generation model fulfils all of these aspects but in a way:

1. Totally integrated, defining in detail the way to accomplish the interfaces among each of the five aspects mentioned. The integration is complete both in the aspect of integration of the best tools for the analysis of the information systems as well as the integration between the IT and business [7]. These two integrations must take into account innovation [16] and added value as two key aspects.
2. Taking into account the positive points of each standard or methodology in existence, avoiding researching in areas where the acquired knowledge is high.
3. Doing a governance model usable by any organization without requiring any adaptations. For this purpose, the standards are simplified taking into account the size of the enterprise and the sector it works for.

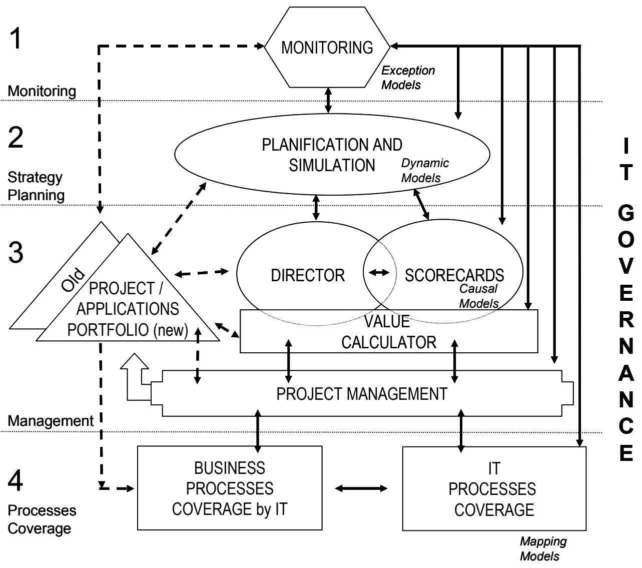


Fig. 1. General view of the Information Model (IG^4).

Within the IG4 model's reach is giving response to the monthly control (real, plan and deviation), to the tacit needs (medium term) and strategy needs (long term). Moreover, the IG4 model starts from a detailed analysis of the business processes coverage by IT and of the processes of any IT department. The said coverage analysis is complemented by an analysis of the double alignment sought (technology/business and strategy/operation) and some control tools. All of these control, coverage analysis, alignment and strategy planning functions are valued and analyzed by means of three useful techniques which are complemented in the IG4 model: metrics, benchmarking and maturity models.

As it has been previously mentioned, there are five key aspects to bear in mind if an efficient and effective governance of the IT is required within the organizations: Audit & control, benchmarking, alignment, strategy planning & simulation and added value techniques. These five aspects are essential when it comes to managing any IT department, so if the governance models do not include them in a totally integrated way:

1. Some key requirements of management or governance are left uncovered.
2. Multiple solutions are forced to be used so the dispersion of the different solutions does not allow the IT's Governance traceability.
3. High efforts of adaptability by the IT departments are demanded so that complete governance tools are available when it is not the CIO (Chief Information Officer) ultimate purpose to prepare or customize these governance tools.

As figure 1 shows, the model boasts four layers that pursue different objectives and where the six system's modules are situated:

1. Monitoring level: it matches up with the monitoring module. It includes key information and alerts of the rest of the modules.
2. Planning level: it matches up with the simulation and planning module. The strategy planning is carried out by means of dynamic models that allow simulation and help the detailed planning that is carried out and supported by the two processes coverage (of business processes and of IT processes).
3. Strategy management and analysis level: This level is covered by two modules: applications portfolio module and the balanced scorecards BSC module (based on the BSC philosophy: Balanced Scorecard). The latter module is divided into the BSC of the business activities coverage by IT and into the BSC of IT activities' coverage. A sub module of project management is incorporated.
4. Processes' coverage level: It details which are the business' processes and what their coverage is by the IT and which are the IT processes and what is the coverage given to these processes. It reaches a level of detail of the processes' activities. The coverage level boasts two information arrangements: by applications with all the supported functionality's details (only to analyze under IT coverage) and by processes to consider, for example, the lead time.

2 Related Work

In a technical codification level there has always been a lack of orientation view to the value, subject that the software's economy discipline has tried to give an answer to [2] [3]. In a more aggregated system's level and not of software the current dispersion in the achievement of governance information is indicated by the different surveys where there is a great dispersion of used methods and there is no massive use tool: IT Balanced scorecards, regulations (ISO 9000, ISO 15000, ISO 17799, etc), governance methods (CobiT, ITIL, Coso, etc), simulation models, project / application based management (PMBOK [18], CMMI [1], etc), quality and process based management (6 SIGMA and Lean [22]), etc. Moreover all of these methodologies are treated in a disintegrated way so that its use with important tools such as applications portfolio and system's strategy planning is complicated.

The start of the IT Governance was due to the classic analytic accountancy conception in which the computing areas were considered centers of structure cost in which some natures of expense where imputed, which by means of sharing out methods proceeded to the assignation to the products or expenses structure in a more or less contrived way. An important advance to this traditional view was brought by the ABC methods (Activity Based Costing) very appropriate for areas with an important level of indirect expenses as it is the computing department's case. This way of distributing the costs by means of some activities allows a sharper assignation (by definition of activities and cost-drivers) of the computing department's expenses to the different units of business, products and accounts of results. Both the ABC method and the traditional method have the same lacks: they do not ensure the alignment with the strategy or businesses, they do not promote or bear the innovation and they do not have at their disposal planning or simulation tools. Also the control and system audits were inexistent as they limited to a few cost centers with six or seven classes of expense.

To the previous insufficient IT governance proposal, continues an interesting approach orientated to ensure the strategy management's monitoring by means of alignment techniques of the main objectives. These methods are based on balanced scorecard of Kaplan and Norton versions with their four views: innovation, internal, client and financial [13]. The IT Balanced Scorecard (itBSC) is a variation of the balanced scorecard based on the typical activities of a computing department summarized in two views, a first one for the acquisition and development and a second view for the delivery and support. The two main points of the IT balanced scorecard are a complete alignment of the IT strategy with the IT operations and a subsequent alignment by means of a cascade of balanced scorecards with businesses. Besides that alignment the balanced scorecards is a good tool for strategy's implementation as it allows the monitoring of the enterprise's objectives once the critical success factors have been defined.

However, the IT balanced scorecard do not give a complete solution for the IT field as they are based on the strategy level, they lack an auditor value and an internal control, and their point of view of ensuring the alignment with the strategy leads them to give an incomplete view and only focus on those chains of critical success factors / goals / indicators meant to be aligned. The theories related to the strategy planning of the information systems cover key aspects such as the decision-taking for the IT's position of our organization in the future. The planning model is key for the IT governance due to the high cost of the investments in applications and the need of recouping during several years [14]. However, these models are not enough for a detailed control of an IT department's different activities so they cannot be used in an isolated way to carry out a correct IT Governance.

A third generation of models and methodologies for the IT governance is brought by an evolution of different IT audit and control methods [10] among which we can find the well-known CobiT [5], Itil [11] and Coso [6] (the latter one with a general approach as for enterprise's audits). These methods, due to their depth, give a great level of detail and have been completed until reaching high levels of rigor and have therefore evolved towards governance models with some peculiarities: Itil focused to the service, CobiT to the control in a more strategy level, ISO17799 if you are looking for a complete security solution [4]. and Coso to the internal business control.

Only taking into account each one's complexity taken separately in addition to the necessity of having the best of each of them leads inexorably to the impossibility of its use bearing in mind the busy agenda of a CIO. Due to the important investments that require software's developments and the application's implementation, the philosophy of the CMMI maturity models must be integrated in the IT governance models [1]. Based on the CMMI philosophy, a very useful tool that complements the governance models are the maturity processes, as they include the time variable and the routes needed to allow a better approach to the benchmarking techniques as it is the appropriate way of weighing up the results without forgetting that the better practices' tools and the standards must be designed in a way that the mere comparison with the best practices does not get to stop the real innovation.
This IG4 method means a new advance to the IT governance by means of a tool that integrates in a forced way the necessary requirements of planning/ simulation/ audit/ control, better practices / added value and alignment. Another aspect that distinguishes the present approach is the existence of an added value module that works like a value calculator of all the modules.

3 General Description of the Model

As it has been mentioned in the previous sections, this IG4 model is a model that pretends to provide solution to the problems of enterprises in the IT field without gathering lots of methodologies or spending a lot of time adapting them. Due to the importance the systems' area has achieved in the enterprise as essential part of survival or as part that sets the difference in the business, the fact of controlling the IT area is not an easy task and it requires the following model described (it is described the three main layers by size reasons):

3.1 Level of IT Strategy Planning

It is in this level where the system's planning to five years is carried out. It is a critical module if considered that the decisions in IT involve important investments to be recouped in large periods of time. Also the IT require simulation tools that allow to analyze different alternatives to determine how current decisions influence (for example selecting a COTS or a CASE tool) in a future costs' decrease (or value generation) in the medium and long term [9][20]. This module is supported by a dynamic system's planning and simulation tool like Vensim or Stella. As the level of detail is to five years, it is lower than that one of modules that supports monthly control information (coverage modules). In this planning and simulation module the information is, because of the tool's needs, of a monthly regularity, whilst in the activities' maps where the planning/real situation/deviation control is carried out in a detailed way the information's level of detail is: in the first future year it is month to month whereas the information's detail in the following four years is four-monthly to four-monthly [21].

The way of feeding this planning and simulation tool is by the coverage modules although only in those concepts that have enough level of detail to boast planning information. The way of calculating the value and costs like the rest of the IG4 model is carried out according to the scheme followed in the value calculating module (section 3.4.). This module sends information to the calculating module and receives the value and costs (as the breakdown that will be indicated) of the calculating module.

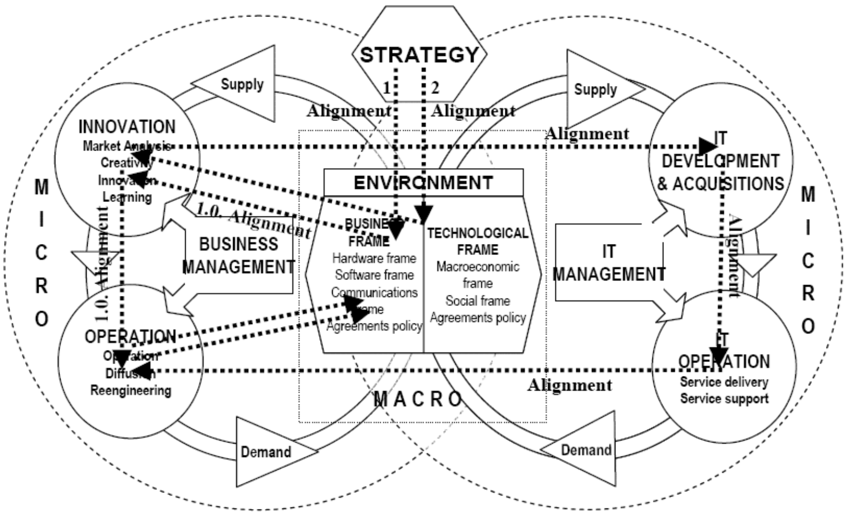


Fig. 2. Balanced Scorecards with active alignment between business and IT.

3.2 Level of Management and Analysis for IT

The Scorecards' module for IT main task is to help defining the strategy, ensure the strategy's fulfilling, ensure the alignment between the business needs and the systems' coverage and analyze the suitability and improvement points of each of the systems and enterprise's applications. In order to achieve these objectives there are two modules: applications portfolio and director scorecards for IT.

1. Applications Portfolio (AP): Both the corporate systems (COTS and legacy) and the systems based on Internet technology are analyzed in this applications portfolio in a level of detail of systems/ modules and sub module (only the main sub modules). The information with a higher lever of detail of the business' processes and its coverage by IT is bore in the processes' coverage modules by IT (it is the module that feeds the applications portfolio).

2. Director Scorecards (DSC): They work similarly to the Balanced Scorecards (BSC) developed by Kaplan and Norton as for causal diagrams but they differ from the latter ones on the fact that they follow a cyclic supply and demand structure that starts with the market's understanding, continues with the innovation followed by the operation and ends as supply in the market again. The IG4 model bears similar treatments at a business and IT level. Taking into account all of what has been said and due to the fact that it is considered more suitable, the financial view of Kaplan and Norton's

BSC is extracted in a calculating module that deals with all the IG4 model. Also as this view depends on the supply/demand, it facilitates a parallelism with the planning simulation that follows a complete model of supply and demand too as it is shown in figure IV for business processes. Just as it has been mentioned the planning view follows a high level of detail but considers all of the business processes and IT processes. On the contrary the BSC specialize in the strategic objectives and all of the steps required in order to define the strategy and ensure the strategy's alignment between businesses and IT.

In the IG4 model the adjustment of the businesses coverage's views by IT and of the IT activities' coverage is innovative. Currently the businesses and systems of information's adjustment is carried out by means of the systems' alignment with businesses in a unique direction that goes form businesses to systems. But this approach implies three problems:

1. A first problem is that in the current enterprises and in sectors such as the bank, a great number of functions and activities totally depend on the information systems: there is no business without IT that bear the business' processes [15]. Any BSC that do not include a complete integration with the IT balanced scorecards are wrong.
2. The second point to take into account is that many business models base their innovation on the technological tools' functionalities so in this case a business alignment is produced simultaneous with the steps indicated by the IT's strategy [8]. The Internet channel in most of the sectors (for example an on-line newspaper or CRM model) are good examples of this tendency in which the technological tools' strategy marks the step of the business' strategy.
3. The client's perspective is not enough and it disintegrates in two scorecards: market analysis (start of the demand) and penetration in the market (end of the supply), which include the key macroeconomic variables of the sector and IT there are at that point of time.

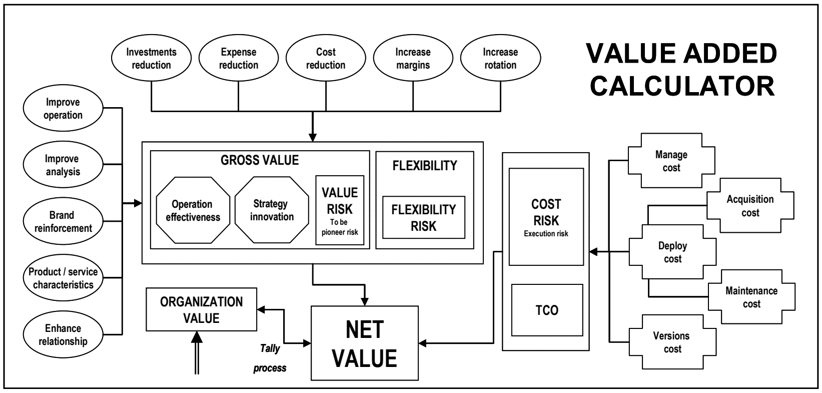


Fig. 3. Value calculator model of the information Governance Model (GI^4).

The alignment is carried out by means of BSC cascades following the businesses' logic and the interrelation with technologies, being possible three routes:

1. When it is about aspects not influenced by the IT and that lead to innovation, the BSC's cascade is the following: markets understanding, businesses innovation, busi-

nesses operation and market penetration. In this case only the monitoring level and the management and analysis level are affected.
2. When it is about aspects not affected by the IT and with no innovation, the BSC's cascade is the following: markets understanding, businesses operation and market penetration.
3. When it is about aspects affected by the IT the DSC's cascade is the following: Markets understanding, businesses / IT innovation (simultaneous), development and IT acquisitions, business / IT operations (simultaneous) and market diffusion.
The third important difference provided by the IG4 model is that the scorecards' financial view has been extracted in a calculating module for several reasons:
1. The calculating formula is fixed and identical, it is required in all of the IG4 model's modules and is not subjected to variations. This calculating formula is based on the strategy innovation concepts, operational efficiency, and some variants of the "Total Cost of Ownership" and "Total Economic Impacts" [23] methods.
2. In this IG4 model, as it has been previously mentioned, the value's management is critical in all the modules as is it the only way of objectifying the IT's improvement points in the organizations that would otherwise remain very subjective. This calculating way is used in the four levels and in the six modules of this IG4 module.

3.3 Level of Processes: Maps and Coverage

This IT governance model has its limits in the Systems of Information's fields. So it is not the IT's governance objective to analyze the business strategy, but, on the contrary, it is this model's objective to analyze the way the information systems give coverage to the business needs detected. Because of this the IG4 model bears two big groups of processes maps: business processes to analyze its coverage by IT and typical processes of the IT departments.
The IG4 model would not be useful to give a complete support of the Computing Field of an organization didn't it support a level of detail enough to justify the decisions and to analyze the source of the inefficiencies. On the contrary, taking into account the IG4 model, the best solution is to reduce the analysis detail because this is an integrated tool with clear executive orientation. For example, reaching a level of detail as exhaustive as the one ITIL can have in the monitoring of the service's management has tried to be avoided (more in the field of daily operation control).
The two large groups of processes and activities bore by the IG4 model are:
1. Coverage map of the business processes by the IT: It follows a supply and demand approach just like BSC (balanced scorecards) and the planning module. The classification followed in order to classify the processes, define their activities and determine the best CSF and metrics (KGI and KPI) has been starting from the main modern theories about businesses. This view also takes for base information an analysis of the magnificent application maps that SAP [19], Navision and Oracle bear distinguishing between basic processes common to the generality of the business sectors and functionality / processes that are particular of specific sectors. The IT portfolio reads this process coverage to group the information according to applications. Besides the most modern theories that manage innovation and operations, in the current decade a good management tool of the IT can't be understood without taking into account the

internal practices of a good finance and social governance (Sarbanes-Oxley [10]) that are also represented within the IG4 model's processes (only to analyze the IT coverage). To allow the testing, each of the processes is analyzed in accordance with the maturity they include: maturity in the operational efficiency, maturity in the strategy innovation, maturity in the risk treatment and maturity in the flexibility treatment. The maturity in the operational efficiency and in the strategy innovation regards the delivery (lead time analysis), the functionality's grade of coverage and the service's quality (confidentiality, integrity, availability, compliance and reliability). As control of the monthly operation this module registers in detail the processes, the critical points of this process, the real data (metrics) and the planning and deviation from the planning for each process. Because of the load of work the control of planning and deviations leads to, this control task is only carried out for the main processes (to make up by the user).
2. Coverage map of the IT processes: Following the same supply/demand scheme used for the business processes, the IG4 model proceeds to analyze the processes and activities of the IT field. Those processes that require so have a detail of system to system. In this module all the information needed for the monthly control and the costs calculation is bore: critical success factors, key performance indicators, key goal indicators and maturity grades. All the costs are accurately allocated to the business processes coverage map.

4 Aspects of the Proposed Model that Make the Difference

As it has been shown throughout this paper, the IG4 model gathers the best of the theories and standards in existence proceeding to include, among others, the following improvements:
1. Integrated and pragmatic approach of the best practices nowadays about IT Governance and Management. Complete and integrated approach that provides solution to the IT governance needs in the organizations of any size by means of a four-level structure: monitoring, strategy planning, management and process coverage. Without a model of IT Governance oriented to services (Service Oriented Enterprise) is impossible to reach high levels of maturity of SOA and SOC (top-down approach).
2. Integration of all the requirements to be regarded in a model of IT Management and Governance by means of six modules: monitoring, planning & simulation, applications portfolio, director scorecards, value calculator and processes coverage. The second forced integration is the integration between the business coverage by IT and the IT activities. This second integration allows defining a Governance of the IT that leads to the investments in IT as springboard for success among organizations.
3. Balanced Scorecards where the finance view, last step in the causal diagram, is replaced by a complete and detailed method of added value that provides service to all of the system's modules at the same time. This added value calculation includes as valuable concepts as benchmarking, metrics, maturity models, Total Economic Impact and Total Cost of Ownership in a tidy way. Also these Balanced Scorecards include alignment routes between business and the closest IT to what really happens in the organizations (for example they start and end in the market).

4. Detailed analysis of the business coverage by the IT with and innovative view of the processes following the supply/demand cycle, what also facilitates a complete parallelism with the strategy planning module. The functionality of strategy planning is totally integrated in the model and it includes a simulation tool too.

References

1. Ahern, D.; Clouse, A. and Turner, R.: CMMI Distilled. Second Edition. A practical to introduction process improvement. Second Edition. Addison-Wesley. Pearson Education (2004).
2. Bakos, J.Y. and Kemerer, C.F: Recent Applications of Economic Theory in Information Technology Research. Decision Support Systems (December 1992).
3. Boehm, B and Sullivan, K.: Software Economics: A Roadmap. Future of Software Engineering. Limerick Ireland. ACM (2000).
4. CobiT: CobiT Mapping. Mapping of ISO/TEC 17799: 2000 with CobiT. IT Governance Institute (2004).
5. CobiT: CobiT 4.0. Control Objectives / Management Guidelines / Maturity Models. IT Governance Institute (2005).
6. COSO: Internal Control over Financial Reporting – Guidance for Smaller Public Companies. Executive Summary (June 2006).
7. Davenport, T.: Putting the Enterprise into the Enterprise Systems. Harvard Business Review (1998).
8. Folgueras, A.; García, A. and Ruiz, B.: A Proposal of Integration between IT Governance and Business Balanced Scorecard. 2007 IRMA International Conference (2007).
9. Forrester, J.W.: Industrial Dynamics. MIT Press, Cambridge, MA (1961).
10. Fox, C. and Zonneveld, P.: IT Control Objectives for Sarbanes-Oxley: The Role of IT in the Design and Implementation of Internal Control Over Financial Reporting, 2nd Edition. Printed in the United States of America. IT Governance Institute (September 2006).
11. itSMF Library: Foundations of IT Service Management based on ITIL. IT Service Management an Introduction, based on ITIL. Van Haren Publishing (2004).
12. itSMF: Planning and Achieving ISO/IEC 20000 Certification. Version 1.0. Office of Government Commerce (OGC) (2006).
13. Kaplan, R.S. and Norton, D.P.: The Balanced Scorecard: Translating strategy into action. Harvard Business School Press (1996).
14. Mintzberg, H.: The Rise and Fall of Strategic Planning. New York: The Free Press (1994).
15. Olazabal, N. G.: Banking the IT Paradox. McKinsey Quarterly. Number 1 (2002).
16. Organization for Economic Cooperation and Development (OECD): A New Economy? The Changing Role of Innovation and Information Technology in Growth (2000).
17. Porter, M.E.: What is Strategy? Harvard Business Review OnPoint (2000).
18. Project Management Institute (PMI): Effective Benchmarking for Project Management (2004).
19. SAP: Solution Componer. Quick Guide. SAP Business Maps & Engagement Tools November 2005.
20. Senge, P.M.: The Fifth Discipline. The Art and Practice of the Learning Organizations (1990).
21. Ward, J. and Peppard, J.: Strategic Planning for Information Systems. Third Edition. John Wiley & Sons, LTD (2002).
22. Womack, J.P. and Jones, D.T.: Lean Thinking. Free Press. First Free Press Edition (2003).
23. Zojwalla, S.: The Total Economic Impact™ Of The Forrester TEI Multicompany Case Study. Forrester Consulting (August 2006).

Technologies

An Algorithm for Automatic Service Composition

Eduardo Silva, Luís Ferreira Pires and Marten van Sinderen

Centre for Telematics and Information Technology, University of Twente
Enschede, the Neterlands
{e.m.g.silva,l.ferreirapires,m.j.vansinderen}@ewi.utwente.nl

Abstract. Telecommunication companies are struggling to provide their users with value-added services. These services are expected to be context-aware, attentive and personalized. Since it is not economically feasible to build services separately by hand for each individual user, service providers are searching for alternatives to automate service creation. The IST-SPICE project aims at developing a platform for the development and deployment of innovative value-added services. In this paper we introduce our algorithm to cope with the task of automatic composition of services. The algorithm considers that every available service is semantically annotated. Based on a user/developer service request a matching service is composed in terms of component services. The composition follows a semantic graph-based approach, on which atomic services are iteratively composed based on services' functional and non-functional properties.

1 Introduction

Advances in mobile communications and devices triggered a multitude of new and innovative services and business areas. For example, value-added services are being proposed by telecommunication companies and service providers, aiming to provide personalized, context-aware and attentive services to end-users. The IST-SPICE (Service Platform for Innovative Communication Environment) project [1] aims at developing a platform to be used by end-users and application developers for the development and deployment of innovative services. SPICE services are composed based on a collection of components, whose services can be published and used in service compositions by end-users and application developers. This is made possible by applying Web Services technology [2] and the Service-Oriented Architecture (SOA) principles [3].

Since it is not economically feasible to build services separately by hand for each individual user, and furthermore, it is hard to predict at design time what personalised services the user may wish, a lot of attention has been given lately to the (semi-) automatic composition of services [9, 10]. Automatic service composition starts with a service request describing the service desired by an end-user or service developer. If there is no service already available that matches the request, a composition of other available services that match the request is constructed. These available services are denoted here as atomic services. In the SPICE project these services are specified in a language called SPATEL (SPice Advanced service description language for TELecommunication services) [4]. This language allows the specification of services in a platform independent manner, including annotations concerning semantic and non-functional properties. Such annotations are imperative in the context of automatic service composition.

One of the activities of the SPICE project is the development of an Automatic Composition Engine (ACE), which should support end-users and application designers on the development of service compositions. The ACE is expected to receive either service requests from end-users in natural language, or from the application designers in some well-defined notation, and deliver a service composition or a list of alternative compositions, respectively. This paper discusses our ideas concerning the automatic service composition algorithm. Our approach relies on the use of semantic annotations on the atomic services, and on service requests, to perform the service discovery, matching and composition. We define our approach as a semantic graph-based automatic composition, where discovered services are represented in a graph, which is used to optimize the search of component services and their composition.

The paper is further organized as follows: section 2 provides some motivation for our work, including a motivating example of application of automatic service composition; section 3 gives an overview of the SPICE Automatic Composition Engine; section 4 presents our initial approach towards the algorithm for automatic service composition; section 5 compares our approach with some related work; and section 6 presents our conclusions and depicts directions for future work.

2 Automatic Service Composition

Automatic service composition aims at automatically composing services that satisfy a given service request from an end-user or service developer. Services are composed in terms of already available atomic services, which are orchestrated in the service composition.

Service requests are used for service discovery, matching and composition. Service requests allow end-users or service developers to specify what they want the service to do for them, abstracting from the way this service is implemented, possibly in terms of a composition of atomic services. In SPICE we are developing the Service Creation Environment, which should create service compositions that support the service requested by an end-user. In order to obtain these compositions automatically, the service request and service descriptions of atomic and composite services need to be annotated with semantics, by using ontologies. Web services [2] are basic building blocks for the realization of services, but they lack semantics. Semantic Web [5] is an effort that provides service descriptions with semantics, which enables automatic reasoning on these descriptions. OWL-S [6] and SPATEL [4] allow the definition and creation of semantic annotated (web) services, using ontologies. These technologies are expected to enable automatic service composition.

A scenario to illustrate automatic service composition is the following: Bob wants to send a happy birthday message in Italian to Monica by SMS. He does not speak Italian so he has to use a dictionary in order to be able to write the message. Imagining that Bob has access to the SPICE platform or another platform that supports automatic service composition, he may issue to the platform the command *send "Happy Birthday my dear!" translated in Italian to +393123456789*. In this case it is highly probable that there is no single service to accomplish this task, so the platform attempts to find an appropriate service composition for that. Two services may have to be used, namely a translator and an SMS messaging service. This process relieves Bob from the hassle of manually discovering each required service and invoking these services.

3 SPICE Automatic Service Composition Engine

The SPICE Automatic Service Composition Engine (ACE) contains four basic components: *Semantic Analyzer*, *Composition Factory*, *Property Aggregator* and *Matcher*. Fig. 1 depicts the ACE architecture.

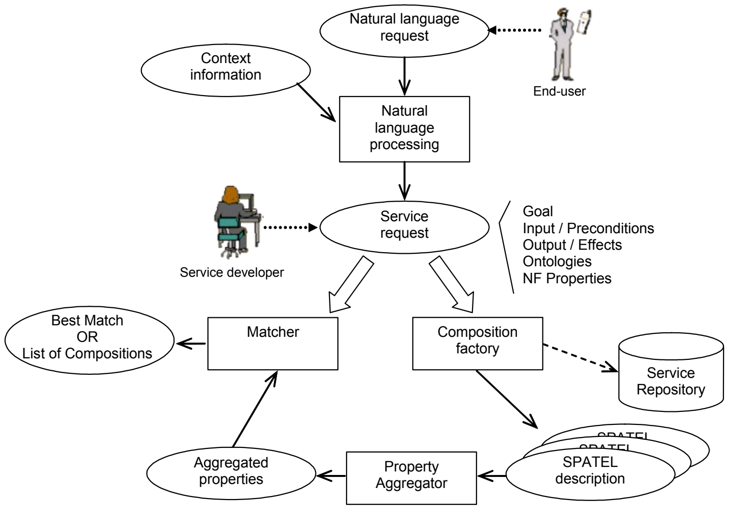


Fig. 1. SPICE ACE architecture.

Fig. 1 represents the two basic ACE usage scenarios: an end-user issues a service request in natural language, and a service developer issues a service request in some well defined formalism. The end-user is shielded from the system's complexity by requesting services in natural language. These requests are processed by the Semantic Analyzer, which constructs a formal service request according to the ACE's service request formalism, which is the same formalism used by the service developer.

When a service request is defined, the Composition Factory queries the service repository for a service that matches this service request. If such a match exists, the matching service is returned. In case no match is found, the Composition Factory creates a composite service that resolves the request. In general the Composition

Factory may generate multiple alternative compositions that match the service request.

Factory may generate multiple alternative compositions that match the service request.

Services and service requests are characterized by their functional and non-functional properties. Functional properties are the services' goals, inputs, outputs, preconditions and effects. These properties are used to perform the service discovery, matching and composition. Examples of non-functional properties are cost, security, performance, reliability, etc. Non-functional properties are used to limit the space of compositions that fulfil the service request, and to rank the generated set of compositions. Service and service request descriptions contain the functional and non-functional properties and the ontologies used to define these properties.

The compositions produced by the Composition Factory are passed to the Property Aggregator component, which computes the non-functional properties of the resulting compositions, by aggregating the non-functional properties of the atomic component services.

The set of generated composite services is then passed to the Matcher component. This component performs a matching between the composed and requested services, using their non-functional properties. In the end-user's use case, the best matching is returned to the end-user. This matching is obtained by taking the user profile and context information into consideration, which are managed by the SPICE platform. For a developer's request, several compositions may be returned. The developer can select the one that best fits his needs, possibly adapting it to fit more specific needs.

4 Semantic Graph-Based Composition of Web Services

The Composition Factory takes a formal request from an end-user or a service developer, and tries to find a service composition that matches the service request. In case a single service that matches the service request already exists, this service is returned as result.

4.1 Algorithm

In ACE, a formal service request contains the following elements: inputs, outputs, preconditions, effects, goals, non-functional properties and a list of domain ontologies. These elements are defined in OWL [15], and are used to discover, match and compose services. Available services are specified in SPATEL, and provided with semantic annotations similar to the elements above. This allows these services to be discovered through the goals of a service request, and then composed with other services by matching their interfaces in terms of inputs, outputs, preconditions and effects. In this paper we omit preconditions and effects in order to simplify the presentation of the algorithm. We expect that the addition of these elements to the algorithm is straightforward, since they can be seen as special cases of inputs and outputs, respectively.

The Composition Factory obtains service compositions by composing services according to a graph-based algorithm. The composition is created using an approach that starts with the outputs and possibly effects, and works backwards in the direction of the inputs and possibly preconditions. This implies that semantic descriptions of goals, inputs and outputs are compulsory for the ACE otherwise service discovery, matching and composition is not possible.

The algorithm holds a set of nodes *N* to be processed. Each element of *N* represents a set outputs *o* and goals *g* that do not match the inputs of the service request. The algorithm starts with a node n_0 representing the outputs and goals of the original service request o_0 and g_0, and issues a query for all the services that provide outputs o_0, support goals g_0 and requires the same inputs as the service request (i_0). In case a service that matches this query is found, the set of nodes *N* becomes empty and the algorithm can stop. In case the query returns a service s_1 that supports part of the goals g_0, delivers outputs o_0 but does not match the inputs i_0, the remaining goals g_1 are stored in the set of nodes *N* with the remaining inputs i_1 of the found service with respect to i_0 indicated as outputs. The algorithm then processes each node n_i by querying the service repository for services that match the goal g_i and the outputs o_i, and either decides to stop on this branch, or add more nodes to *N*, depending on the result of the query.

Fig. 2 shows the steps of our composition algorithm.

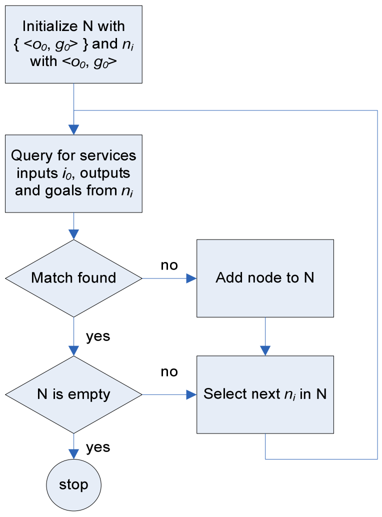

Initialize N with { <o_0, g_0> } and n_i with <o_0, g_0>
Query for services inputs i_0, outputs and goals from n_i
Match found
no
Add node to N
yes
N is empty
no
Select next n_i in N
yes
stop

Fig. 2. Service composition algorithm.

The algorithm delivers a composition graph as result, with possibly several branches representing alternative service compositions going from the requested inputs i_0 to the requested outputs o_0 and covering the requested goals g_0. The algorithm can execute indefinitely if matches are not found, or may result in compositions of too many component services. This implies that some heuristics must be defined to limit the algorithm execution and yet deliver useful results. Some possibilities are:

– stop at some graph depth, i.e., when the branches reach a certain number of edges. Since branches represent compositions, these heuristics limit the number of services in a composition;
– use non-functional properties of the composed service to select compositions that comply to the service request. This requires the calculation of the non-functional properties of the composed services, which can be done by aggregating the non-functional properties of the component services. By selecting only compositions that comply with non-functional properties, again the number of services in a composition is limited.

A measure of *semantic similarity* [7, 8] can also be used to determine the semantic distance between the query and the resulting services. This allows the algorithm to generate composition graphs with alternative semantically close branches, which can be useful in case no perfect match is found.

4.2 Example

We present an example to illustrate our automatic service composition algorithm. Considering the example in which Bob wants to send an SMS message in Italian to Monica (see section 2). Bob specifies its request in natural language as: *send "Happy Birthday my dear!" translated in Italian to +393123456789*. Since Bob is using the SPICE platform, a concrete service composition is going to be constructed at runtime, in order to cope with Bob's natural language service request. The ACE's Semantic Analyser extracts the desired service properties, creates a formal service request and passes it to the Composition Factory component. The formal service request can be represented as:

```
<Input>
        <"Translation:Language" name="sourceLanguage">
        <"Translation:Language" name="targetLanguage">
        <"Translation:Text" name="textToTranslate">
        <"Mobile:Telephone" name="destNumber">
</Input>
<Output/>
<Goal>
        <"Goal:translate">
        <"Goal:sendSMS">
</Goal>
<Non-functional>
        <"Latency:Response" value=5>
</Non-functional>
<Ontologies>
        <Goal Mobile Latency Translation IOTypes>
</Ontologies>
```

Fig.3 shows part of the Translation ontology that we assume in this example.

The Composition Factory takes this request and queries for a service that matches the inputs, outputs and goals. Suppose no service matches these elements, but the query returns the service with a sendSMS goal, matching `Goal:sendSMS` and no outputs, matching the outputs of the original service request. The `sendSMS` service is added to the composition graph, and a node representing the inputs of this service and the remaining (unsolved) goal is added to the set of nodes *N*. The inputs of the sendSMS service are destNumber and the text message to be sent. destNumber is of type Mobile:Telephone and corresponds to the telephone number given in the natural language request. The text message does not match the original input text of the service request since it is not in Italian. This means that a service with goal `Goal:translate` is necessary to provide an output of the type Translation:Text that is in Italian. Fig. 3, shows an excerpt of a Translation ontology that can be defined for this example. This ontology relates `Text` to the Language in which it is written. The translation service (not explicitly specified here) can translate text in one (source) language to text written in another (target) language. In this concrete example the translation is from English to Italian, which corresponds to the sourceLanguage and targetLanguage, respectively.

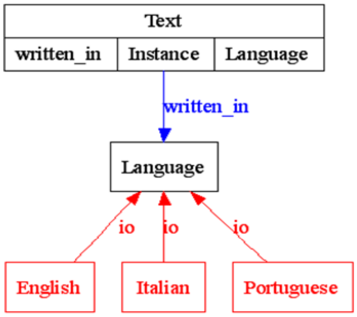


Fig. 3. Excerpt of the Translation ontology for our example.

We assume that the Translation service is found as a result of querying for a service that supports the `Goal:translate` goal and supports English and Italian as inputs for sourceLanguage and targetLanguage (the current node n_i). In this way the inputs of the original service request are completely resolved, and the composition process can stop, resulting on a composition of the services Translation and sendSMS.

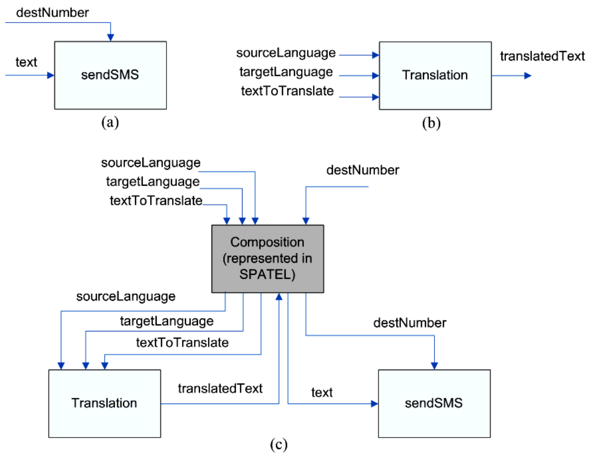


Fig. 4. Service composition example.

Fig. 4 depicts the composition process in terms of the services obtained throughout the process. Fig. 4 (c) shows the resulting composition. In SPICE, compositions are represented as SPATEL specifications, which could be translated to BPEL or executed directly by a SPATEL engine (currently being developed).

5 Related Work

In this section we present a brief overview of some techniques that cope with automatic service composition. For a detailed survey we refer to [9, 10]. We consider two techniques, namely graph-based and interface-matching automatic composition.

In [11], the authors propose a graph-based approach that constructs a composite service out of atomic services in case no single atomic service can satisfy a request. OWL-S [6] is used to describe the web services, in terms of inputs, outputs and execution workflow. A formalism and modelling tool called "interface automata" [12] is used to represent web services' information and perform compositions. Atomic services are stored in the repository as a graph, where nodes represent input and output parameters and edges represent web services. Each web service contains a description of its inputs, outputs and a dependency set of other web services. Given the graph, the previous information is used to discover the compositions that satisfy the request. The compositions can be constructed using four basic operations, namely concatenation, conditional structure, parallel structure and loop structure. When the compositions that match the request are discovered on the graph, they are passed to the interface automata tool, which performs the composition of the service based on the defined operation structures. If several alternative compositions are found, no mechanism for optimal selection is provided. There are no stop conditions either, which may complicate the search when several compositions do match the request.

The Interface-Matching Automatic (IMA) composition [13] aims at the generation of composite services by capturing expected service outcomes when a set of inputs is provided by the user. The result is a sequence of atomic services, whose combined execution achieves the user goals. Semantic web techniques are used to specify service semantics. Terms and concepts such as inputs, outputs and goals are described using the DAML-S service ontology [14]. Having this representation it is possible to proceed to the construction of composite services. The IMA service composition technique extracts inputs, outputs and constraints from the user request and navigates through the ontology to find the service sequences that match the user's input. After that, it chains services until they deliver the expected output. The goal is to find a composition that produces the best match within the shortest path in the graph, by using the notion of semantic similarity as matching metrics. Our proposed algorithm can be considered as an extension of this algorithm for what concerns the use of the non-functional constraints to select compositions.

6 Conclusions and Future Work

In this paper we present our initial proposal for the automatic service composition algorithm of the SPICE project. Fully automated composition is still an open research issue, and not many concrete results have been achieved in this area yet. By assigning semantic annotations to services, reasoners can be applied to automate the composition process to some extent. The application of these techniques to solve realistic service composition problems is yet to be assessed.

Our approach is based on semantic graph composition, and considers that service requests and service descriptions define functional and non-functional properties of required and offered services, respectively. We propose an algorithm for the Composition Factory that uses service goals and input/output matching to perform service composition. We assume the availability of a repository, organized according to domain ontologies. The Composition Factory is in principle capable of creating different alternative compositions that match the service request. The algorithm has been explained and a simple example has been used to illustrate its basic operation.

Our algorithm is still under development, and some improvements are expected to take place. Once we have a prototype, we expect to explore and improve several points of our algorithm, namely: the optimization of the composition process, the use of similarity measures in case no composition fulfils completely a service request; and the possible storage of service compositions created before for using in new compositions.

Acknowledgements

This work is supported by the European IST-SPICE project (IST-027617) and Dutch Freeband A-MUSE (Architectural Modelling for Service Enabling in Freeband) project (BSIK 03025).

References

1. Christophe Cordier et al.: Addressing the Challenges of Beyond 3G Service Delivery: the SPICE Platform. In: Workshop on Applications and Services in Wireless Networks (ASWN'2006). May 2006.
2. D. Booth et al.: Web Services Architecture. http://www.w3.org/TR/ws-arch, W3C Workinggroup Note. February 2004.
3. T. Erl: Service-Oriented Architecture (SOA): Concepts, Technology and Design. Prentice Hall, 2005.
4. J. P. Almeida, A. Baravaglio, M. Belaunde, P. Falcarin, E. Kovacs: Service Creation in the SPICE Service Platform. In: Wireless World Research Forum meeting on "Serving and Managing users in a heterogeneous environment", November 2006.
5. Semantic Web, http://w3.org/2001/sw.
6. David Martin et al.: OWL-S: Semantic Markup for Web Services. http://w3.org/Submission/OWL-S. November 2004.

7. K. Fujii and T. Suda: Dynamic service composition using semantic information. In: International Conference on Service Oriented Computing (ICSOC'04), November 2004, pp. 39-48.
8. F. Lécué, A. Léger: Semantic Web Service Composition Based on a Closed World Assumption. In: European Conference on Web Services (ECOWS'04), December 2006, pp. 233-242.
9. A. Alamri et al.: Classification of the state-of-the-art dynamic web services composition. In: International Journal of Web and Grid Services 2006 - Vol. 2, pp. 148-166.
10. J. Rao, X. Su: A Survey of Automated Web Service Composition Methods. In: Semantic Web Services and Web Process Composition (SWSWPC'04), July 2004, pp. 43-54.
11. S. V. Hashemian and F. Mavaddat: A Graph-Based Approach to Web Services Composition. In: Symposium on Applications and the Internet (SAINT'05), January 2005, pp. 183-189.
12. L. Alfaro and T. Henzinger: Interface automata. In: Symposium on Foundations of Software Engineering (FSE'2001), September 2001, pp. 109-120.
13. R. Zhang et al., Automatic Composition of Semantic Web Services. In: International Conference on Web Services (ICWS '03), June 2003, pp. 38-41.
14. A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, et. al.: DAML-S: WS Description for the Semantic Web. In: International Semantic Web Conference (ISWC '02), June 2002, pp. 348-363.
15. M. K. Smith, C. Welty, D. L. McGuinness: OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/.

Interoperating Context Discovery Mechanisms*

Tom Broens[1], Remco Poortinga[2] and Jasper Aarts[1]

[1]Centre for Telematics and Information Technology, ASNA group, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
http://asna.ewi.utwente.nl, t.h.f.broens@utwente.nl, j.e.aarts@student.utwente.nl
[2]Telematica Instituut, PO Box 589, 7500 AN Enschede, The Netherlands
http://www.telin.nl, remco.poortinga@telin.nl

Abstract. Context-Aware applications adapt their behaviour to the current situation of the user. This information, for instance user location and user availability, is called context information. Context is delivered by distributed context sources that need to be discovered before they can be used to retrieve context. Currently, multiple context discovery mechanisms exist, exhibiting heterogeneous capabilities (e.g. communication mechanisms, and data formats), which can be available to context-aware applications at arbitrary moments during the application's lifespan. In this paper, we discuss a middleware mechanism that enables a (mobile) context-aware application to interoperate transparently with different context discovery mechanisms available at run-time. The goal of the proposed mechanism is to hide the heterogeneity and availability of context discovery mechanisms for context-aware applications, thereby facilitating their development.

1 Introduction

The Service-Oriented Architecture (SOA) paradigm provides a promising approach to develop distributed applications. In this paper, we are concerned with (distributed) context-aware applications, which are applications that use the current situation, called context, to adapt their behaviour [1]. There are numerous examples of possible types of context, depending on the goal of the application. Examples of context are user location and availability, room temperature, and available bandwidth on a communication link. Context-aware applications use information on the context to adapt their functionality with the aim of improving the quality of the service offered to the user. For example, a context-aware 'buddy navigation application' that can offer quick and personalized navigation to available buddies, based on the location of the user and his buddies, correlated with the availability of the buddies. Context is usually provided by various distributed context sources (e.g. GPS sensors for location information, calendar for scheduling or availability information, MSN messenger for buddy information, weather stations for current weather conditions). The application

* This work was partly supported by the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. Additionally, this work was partly supported by the European Commission as part of the IST-IP AMIGO project under contract IST–004182.

environment may contain several useful context sources at any point in time, however, due to for example the mobility of the application or the context sources, the number and identity of context sources may change over time. This requires mechanisms to discover context sources before an application can retrieve context information. Currently there is a trend towards middleware mechanisms that facilitate the development of context-aware applications [2]. Major contributions in this area are context management systems that facilitate the context exchange process, including, amongst others, the discovery of context sources. Consequently a vast amount of context discovery mechanisms exist, which have different capabilities and scope [2-5]. We believe it is unlikely that there will be one future commonly adopted context discovery mechanism. As implied by the diversity of currently available context discovery mechanisms, different application environments (e.g. ad-hoc environments, telco environments) require different mechanisms to exchange their context information. Therefore, the mechanisms, which context-aware applications have to use to discover context sources from these environments, will be diverse. Consequently, (mobile) context-aware applications are likely to be exposed to multiple and changing context discovery mechanisms during their lifespan. Without supporting mechanisms to cope with this aspect, developers have to design and incorporate interoperability mechanisms or consider every possible mechanism statically in their application. Besides the required, substantial, programming effort, this also distracts from the primary task of developing context-aware applications.

In our view, there are three approaches to interoperate context discovery mechanisms:

> Standardisation: every environment that wants to offer context discovery uses one standard context discovery mechanism. However, as already indicated, due to the heterogeneity and different requirements of the application environments, this is not feasible or likely.
> *Bridging:* every environment has a different discovery mechanism that is internally bridged to other discovery mechanisms by bridging components or code.
> *Homogenising:* every environment has different discovery mechanisms that are homogenized by a generic middleware layer, optionally co-located with the application (see Figure 1 for a comparison of the bridging and homogenising approach).

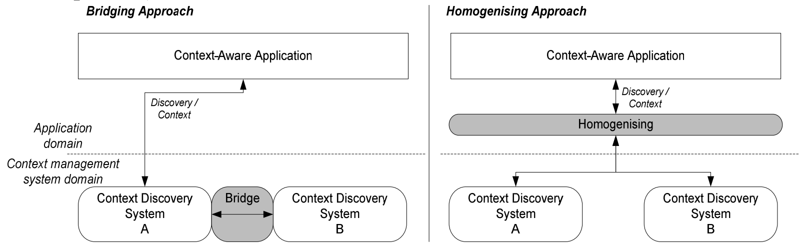

Bridging Approach
Context-Aware Application
Discovery / Context
Application domain
Context management system domain
Context Discovery System A
Bridge
Context Discovery System B
Homogenising Approach
Context-Aware Application
Discovery / Context
Homogenising
Context Discovery System A
Context Discovery System B

Fig. 1. Comparison of the bridging and homogenising approach.

In this paper, we explore the homogenising approach and propose a middleware mechanism that enables a context-aware application to interoperate transparently with different context discovery mechanisms that are available at run-time. The goal of the proposed mechanism is to hide the heterogeneity and availability of context discovery mechanisms for context-aware applications, thereby facilitating their development. We envision the proposed homogenising approach as part of a comprehensive SOA infrastructure to support composable context-aware services. The AWARENESS project (http://awareness.freeband.nl) also explores the bridging approach which is discussed in [3, 6]. The remainder of this paper is structured as follows: section 2 discusses a motivating scenario of a context-aware application that uses multiple context discovery mechanisms. Section 3 presents an analysis of current context discovery mechanisms. This analysis is used to derive requirements for our interoperability mechanism. Section 4 discusses our contribution towards interoperability of context discovery mechanisms. Section 5 presents our proof-of-concept prototype and gives an initial qualitative evaluation. Finally, in section 6, we will end with some conclusions.

2 Scenario

In this section, we reconsider the context-aware 'buddy navigation application' and use it to identify key difficulties that application developers face when developing context-aware applications.

Dennis is a young adult, always wanting to be in contact with his friends. He has a mobile device running the 'buddy navigation application'. This application is able to navigate to available buddies by using location and availability context information of him and his friends. Dennis notices that Monica is in the mall and available for a cup of coffee. He decides to visit her. He instructs the 'buddy navigation application' to help him find her. Inside Dennis' home, a RFID based location context source, found by his home context discovery mechanism, provides accurate location of Dennis. From Monica no precise location source is available in Denis's home, it is only known that she is somewhere in the mall. The 'buddy navigation application' instructs Dennis to take the car to the mall. When Dennis leaves his home, to go on his way to Monica, his home discovery mechanism becomes unavailable. The application switches to a cell based location context source found by the context discovery mechanism of his telecommunication provider. On entering the mall Monica is in, accurate context information on Monica's location becomes available, offered by a Bluetooth beacon context source found by the context discovery mechanisms of the mall. The buddy navigation application pops up a map of the mall, to instruct Dennis how to walk to the book store where Monica is currently shopping.

From the scenario, we can identify the following difficulties, related to context discovery that application developers face when developing context-aware applications:

Finding of context sources through different context discovery mechanisms (e.g. 'home', 'telecommunication', and 'mall' context discovery mechanisms).

Fluctuating availability of context discovery mechanisms (e.g. when Dennis leaves his home his home context discovery mechanism is not available any more).

In this paper, we propose a middleware mechanism that has the goal to support context-aware applications to interoperate with multiple heterogeneous context discovery mechanisms, considering their availability during the lifetime of the application.

3 Analysis of Current Context Discovery Mechanisms

As indicated, many context discovery mechanisms exist. We analysed a subset of these mechanisms consisting of four approaches developed in the AWARENESS project (CMF, CCS, CDF, JEXCI)[3], and one approach developed in the AMIGO project (CMS)[7] and four external approaches (Context Toolkit [5], PACE [2], Solar [8], and JCAF[4]). The result of our analysis is presented in Table 1. The analysis consisted of reviewing the following aspects of the different discovery mechanisms:

<u>*Interaction mechanism:*</u> what interaction mechanisms do the analyzed discovery mechanisms support?

<u>*Interaction syntax:*</u> what information is expressed in the context discovery request?

<u>*Interaction format:*</u> what is the data format of the request and response?

Table 1. Context discovery mechanisms analysis results.

	Interaction mechansism		**Interaction syntax**			**Interaction format**
Frameworks	Req-Resp	Sub-Not	Entity	Type	QoC	
CMF	v	v	v	v	v	RDF
CCS	v	v	v	v	v	SQL/PIDF
CDF	v	v	v	v	v	RDF/PIDF
Jexci	v	v	v	v	v	*Negotiable (PIDF/java objects)*
CMS	v	v	v	v	v	RDF
Context Toolkit	v	v	v	v	-	XML
Pace	v	v	v	v	v	Context Modelling Language
Solar	v	v	v	v	-	?
JCAF	v	v	v	v	-	Java objects

We distinguished the following common aspects in the analysed approaches, which pose requirements on the capabilities our interoperability mechanism should offer to context-aware applications it supports:

The 'request-response' and 'subscribe-notify' interaction mechanisms are offered.

Requests for context minimally specify an entity and context type (e.g. 'Location' of 'Tom').

Requests for context may contain Quality of Context [9, 10] criteria.

We distinguished the following uncommon aspects in the analysed approaches, which pose requirements on the heterogeneity our interoperability mechanisms should have to overcome:

Data models (syntax and semantics) of the request and response (ontology, simple strings, binary).

Communication technologies (e.g. web services, jini).

Intelligence inside the context discovery mechanism (e.g. reasoning, selection).

4 Context Discovery Interoperability Mechanism

The scenario in section 2 and the analysis in section 3 indicate the type of difficulties a context-aware application may typically encounter. A viable context discovery interoperability mechanism will have to hide these difficulties or, at least, diminish the burden placed on the context aware application and its developer to overcome these difficulties.

Summarizing, the problems such an interoperability mechanism has to solve can roughly be divided into the following categories:

(Un)availability of context discovery mechanisms.
Heterogeneous interaction behaviour and communication mechanisms.
Heterogeneous data syntax and semantics.

In this paper, we mainly focus on the first two aspects and syntactic interoperability. We acknowledge the importance of having both syntactic and semantic interoperability; however, this is out of the scope of this paper and we consider this future work.

The pivotal requirement is the ability for a context discovery interoperability mechanism to dynamically adapt to different context discovery mechanisms appearing and disappearing. Based on the knowledge of which context discovery mechanisms are currently available, the interoperability mechanism should then change the way it discovers and retrieves context information on behalf of the applications it supports. By concentrating the specific functionality of the specific discovery mechanism in individual components that can be loaded and unloaded dynamically, the interoperability mechanism does not need to support all separate discovery mechanism simultaneously, and at the same time it is able to abstract from the specifics of individual discovery mechanisms, since this is hidden in the components itself. We call these environment specific components *context discovery adapters* (see Figure 2). For storing and retrieving these adapters at run-time, an *adapter supplier* is defined, which is a repository for discovery adapters. By allowing multiple adapter suppliers to co-exist (e.g. multiple environments), and not prescribing where these suppliers should be located (i.e. not restrict a repository to be co-located on the same host running the context-aware application), specific context discovery environments may provide their own adapter supplier, without losing the ability to use preferred adapters present in the co-located repository. The remaining item to be addressed is the monitoring of the availability of known context discovery mechanisms. Analogous to environment specific adapters, environment specific *monitors* are defined, which are responsible for detecting whether a particular context discovery mechanism is currently (still) available. Adapter suppliers present in a context discovery environment also provide these monitors; next to the discovery adapters mentioned earlier. The adapter supplier co-located on the same host as the context aware application also provides monitors for a set of predefined context discovery mechanisms.

An adapter supplier thus has the following responsibilities:

Provide adapters for the specific context discovery mechanism within its environment.
Provide monitors for the same specific context discovery mechanism, which allow the context discovery interoperability mechanism to detect its availability.

The latter responsibility of the adapter supplier also implies that for the first detection of context discovery mechanisms that can be used with the interoperability mechanism, it is sufficient to detect the presence of an adapter supplier. In order to leverage this approach, rather than creating a new discovery problem, the method to discover such a supplier should be standardised.Next to the different components, additional logic is necessary for the orchestration of the different adapters, monitors, and suppliers. This logic is provided by the *Discovery Coordinator*.

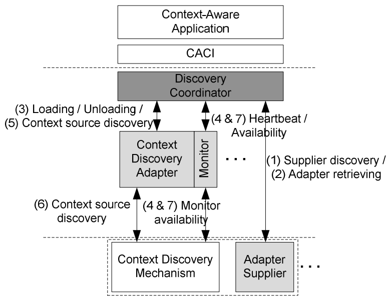

Context-Aware Application
CACI
Discovery Coordinator
(3) Loading / Unloading /
(5) Context source discovery
(4 & 7) Heartbeat / Availability
Context Discovery Adapter
Monitor
(1) Supplier discovery /
(2) Adapter retrieving
(6) Context source discovery
(4 & 7) Monitor availability
Context Discovery Mechanism
Adapter Supplier

Fig. 2. High-level design of the proposed Context Discovery Interoperability mechanism.

The proposed discovery interoperability mechanism is part of a more comprehensive effort towards a context binding transparency [11]. This transparency hides the complexity from the application developer of discovering, selecting, and binding to context sources, which he requires for his context-aware application. Furthermore, it maintains the binding with a context source, thereby coping with their dynamic availability. All these responsibilities are shifted to a middleware layer, coined CACI, which is co-located with the application. For more information on CACI see [12, 13]. Therefore, we integrated the presented interoperability mechanism with the CACI middleware. A typical scenario of the use of the discovery interoperability mechanism is as follows (see Figure 2 and 3): on start-up of the application and CACI, the Discovery Coordinator initiates a discovery of available adapter suppliers (1); this is done continuously e.g. by passive service discovery. When a supplier is found its available adapter/monitor combinations are downloaded (2). The monitor is started (3) to check the availability of the discovery mechanism (4). If it is indeed available, then the corresponding adapter is registered to the Discovery Coordinator, and passed on to CACI, which in turn will use the adapter to discover context sources (5 & 6). The monitor is continuously keeping track of the availability of the discovery mechanism it supports (7). If discovery mechanisms become unavailable, their adapters are deregistered with the coordinator, and indirectly with CACI. Although the figures suggest that only one monitor and adapter is active, multiple monitors and adapters can co-exist at the same time and can become active or inactive during the lifespan of the application.

5 Implementation

Summarising, the architecture contains the following components with their respective responsibilities:

> *Context Discovery Adapter:* component that translates between a specific context discovery framework and a context-aware application in the form of the CACI layer.
> *Monitor:* component that keeps track of the availability of a specific context discovery mechanism.
> *Adapter Supplier:* component that provides the Context Discovery Adapter and the Monitor to the Discovery Coordinator.
> *Discovery Coordinator:* component that orchestrates the interactions between the different components.

We made a proof-of-concept implementation of the discovery interoperability mechanism using an implementation of the OSGi component framework specification. OSGi implementations offer ‘a service-oriented, component-based environment for developers and offers standardized ways to manage the software lifecycle’ (see http://www.osgi.org). The open source OSGi implementation ‘Oscar’ was used as the basic implementation platform (http://oscar.objectweb.org). However, the prototype is also tested on the Knopflerfish OSGi (http://www.knopflerfish.org) implementation. For communication and discovery mechanisms the middleware of the IST Amigo project (http://www.amigo-project.org) was used, which amongst other things, provides components for easy Web Service communication (for both server and client side) and Web Services Dynamic Discovery (WS-Discovery), which uses multicast to discover web services of specific type and scope in the network. More specifically, the WS-Discovery mechanism, available from the Amigo project, was used as the ‘standard’ discovery mechanism for finding adapter suppliers. Every component in the architecture was implemented as a separate OSGi bundle (the OSGi name for a component), which has the added benefit that the bundle id can be used for identifying component instances. The OSGi framework is service-oriented and also deploys the concept of service listeners, which means that components can register themselves as being interested in services of a specific type. If a component that offers a specific type of service is installed and activated, all interested service listeners will be informed of that event. In the prototype implementation the service listener pattern is used by CACI to get notified whenever new Context Discovery Adapters become active after being downloaded by the discovery coordinator. A sequence diagram (see Figure 3) will help to derive the detailed functions of the different components and their respective interfaces. In the text below, italic text in brackets will indicate the interface name that is relevant for the mentioned interaction. In order to be discoverable by a Discovery Coordinator, an Adapter Supplier registers itself with a scope of ‘urn:CADC’ and a service type of ‘IAdapterSupplier’. After an adapter supplier is discovered, the Discovery Coordinator needs to retrieve the list of components provided by the adapter supplier (*IAdapterSupplier*), typically consisting of one Monitor and one or more Context Discovery Adapters. The Discovery Coordinator will download (using OSGi’s component downloading capabilities via http or file system) the components returned by the Adapter Supplier and start the Monitor by activating the Monitor component. If the Monitor successfully detects the context discovery mechanism supported by the adapters, it will start the adapter(s) and indicate the availability of the context discovery mechanism to the Discovery Coordinator (*IDiscoveryCoordinator*). The CACI framework will be notified of this since the started adapters provide a specific service, for which CACI has registered as a service listener. CACI will call the Context Discovery Adapters for performing Context Source Discovery (*IDicoveryAdapter*). The Monitor will keep checking the availability of its Context Discovery Mechanism. If it becomes unavailable, the Monitor will inform the Discovery Coordinator (*IDiscoveryCoordinator*) by deregistering and stopping the relevant adapters. Since stopping the adapters automatically means that the OSGi service they are offering disappears, CACI (as a service listener) will be notified of this event by the OSGi framework. Next to the Monitor, the Discovery Coordinator will continuously check for the availability of the Adapter Supplier via a straightforward heartbeat mechanism; essentially a dummy method call on the Adapter Supplier (*IAdapterSupplier*). If the supplier becomes unavailable, the coordinator will respond by stopping the Monitor belonging to the supplier that disappeared (*IMonitor*).

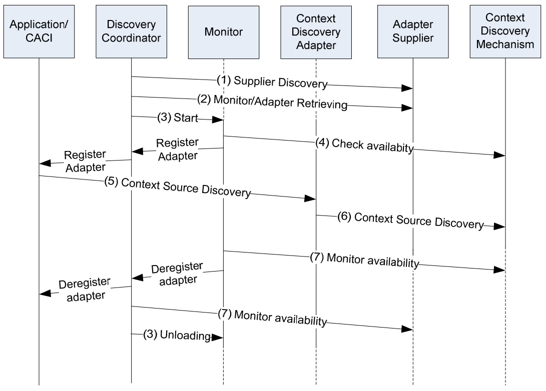


Fig. 3. Time-sequence of a scenario of loading and unloading of a context discovery adapter.

The following code snippets give interface definitions in pseudo-code of the different described components. These where already referred to in the text above.

```
IAdapterSupplier
String heartBeat(void)
Adapter[] listAdapters(void)
URL getAdapterReference(adapterID)

IDiscoveryAdapter
String getFriendlyName(void)
[] discoverContextProducers(request)

IMonitor
String getFriendlyName(void)
Long getComponentID(void)
IDiscoveryCoordinator
newMonitor(IMonitor)
monitorGone(IMonitor)
frameworkAvailable(IMonitor)
frameworkGone(IMonitor)
```

Implementations of the Adapter and Monitor components were made for the CCS, CMS, [3], and SimuContext [14] context management frameworks. They are currently being evaluated. For supporting other discovery mechanisms than the ones already implemented for the prototype, new Monitors and Adapters have to be developed. Since a large part of the functionality of the Monitor is equal for every type of context discovery mechanism, a new one can be implemented by deriving from the reference monitor component and implementing the template parts for the specific needs of the targeted context discovery mechanism. The specific Context Discovery Adapters are less generic than the Monitor, and should at least implement the IDiscoveryAdapter interface. The Discovery Coordinator and Adapter Supplier are generic and do not have to be (re-) implemented for new context discovery mechanisms, although the Adapter Supplier will have to be configured with the appropriate information for the context discovery mechanism it has to support (i.e. URLs of monitor and adapters).

6 Conclusions

This paper discusses work in progress towards a context discovery and delivery interoperability mechanism. In this paper, we focus mainly on the interoperability of context discovery mechanisms. By using the context discovery interoperability mechanism, application developers are relieved from programming mechanisms in their application to interoperate with different context discovery mechanisms that can appear and disappear at arbitrary moments during the life-span of the application. The mechanism actively searches for context discovery mechanisms and when found adds them to the scope of the interoperability mechanism by downloading discovery adapters made available by the discovery mechanisms. Furthermore, it continuously monitors the availability of discovery mechanisms and if one disappears, it removes the adapter from the interoperability mechanism. However, we acknowledge some aspects that are not addressed in this paper and which we consider possible future research:

<u>*Interoperable context delivery:*</u> this paper focuses on the first step of the context discovery and delivery process, namely context (source) discovery. After context discovery, the actual context has to be transferred from the context source to the application, which poses a similar interoperability issue. The chosen dynamic adapter-based approach is designed for both discovery and delivery of context. However, in this paper the approach is only detailed for interoperating context discovery mechanisms. Therefore, we plan to further extend this mechanism with context delivery interoperability.

<u>*Security:*</u> downloading ‘unknown’ code is considered a security risk. However, mechanisms exist to overcome this issue, such as code signing and firewalling [15].

<u>*Semantic interoperability:*</u> in this paper, we focus on functional interoperability. However, interoperating the different data models used by the context discovery mechanisms is similarly important. Mechanisms exist to get semantic

interoperability, which could be used to extend the current mechanism (e.g. ontologies [16]).

References

1. Dey, A., Providing Architectural Support for Context-Aware applications. 2000, Georgia Institute of Technology.
2. Henricksen, K., et al., Middleware for Distributed Context-Aware Systems, in DOA 2005. 2005, Springer Verlag: Agia Napa, Cyprus.
3. Benz, H., et al., Context Discovery and Exchange, in Freeband AWARENESS Dn2.1, P. Pawar and J. Brok, Editors. Freeband AWARENESS Dn2.1, 2006.
4. Bardram, J., The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications, in Pervasive Computing. 2005: Munchen, Germany.
5. Dey, A., The Context Toolkit: Aiding the Development of Context-Aware Applications, in Workshop on Software Engineering for Wearable and Pervasive Computing. 2000: Limerick, Ireland.
6. Hesselman, C., et al. Interoperating Context Managment Systems for Pervasive Computing Environments. in forthcomming. 2007.
7. Ramparany, F., et al. An Open Context Management Information Management Infrastructure. in Intelligent Environments (IE'07). 2007. Ulm, Germany.
8. Chen, G. and D. Kotz. Solar: An open platform for context-aware mobile applications. in International Conference on Pervasive Computing. 2002. Zurich, Zwitserland.
9. Buchholz, T., A. Kupper, and M. Schiffers. Quality of Context: What it is and why we need it. in 10th Workshop of the HP OpenView University Association (HPOVUA03). 2003. Geneva, Switzerland.
10. Sheikh, K., M. Wegdam, and M.v. Sinderen. Middleware Support for Quality of Context in Pervasive Context-Aware Systems. in IEEE International Workshop on Middleware Support for Pervasive Computing (PerWare'07). 2007. New York, USA.
11. Broens, T., D. Quartel, and M.v. Sinderen. Towards a Context Binding Transparency. in 13th EUNICE Open European Summer School. 2007. Enschede, the Netherlands: Springer.
12. Broens, T., et al., Dynamic Context Bindings in Pervasive Middleware, in Middleware Support for Pervasive Computing Workshop (PerWare'07). 2007: White Plains, USA.
13. Broens, T., A. Halteren, and M.v. Sinderen. Infrastructural Support for Dynamic Context Bindings. in 1st European Conference on Smart Sensing and Context (EuroSSc'06). 2006. Enschede, the Netherlands.
14. Broens, T. and A. van Halteren. SimuContext: simulating context sources for context-aware applications. in Intl. Conference on Networking and Services (ICNS06). 2006. Silicon Valley, USA.
15. Rubin, A.D. and D.E. Geer, Jr., Mobile Code Security. IEEE Internet Computing, 1998. 2(6): p. 30-34.
16. Blackstock, M., R. Lea, and C. Krasic. Towards Wide Area Interaction with Ubiquitous Computing Environments. in 1st European Conference on Smart Sensing and Context (EuroSSc'06). 2006. Enschede, the Netherlands.

Using Temporal Business Rules to Synthesize Service Composition Process Models

Jian Yu[1], Jun Han[2], Paolo Falcarin[1] and Maurizio Morisio[1]

[1] Department of Automation and Information, Politecnico di Torino, 10129 Turin, Italy
{jian.yu, paolo.falcarin, maurizio.morisio}@polito.it
[2] Faculty of ICT, Swinburne University of Technology, 3122 Hawthorn, Australia
jhan@ict.swin.edu.au

Abstract. Based on our previous work on the conformance verification of service compositions, in this paper we present a framework and associated techniques to generate the process models of a service composition from a set of temporal business rules. Dedicated techniques including path-finding, branch structure introduction, and parallel structure introduction are used to semi-automatically synthesize the process models from the semantics-equivalent Finite State Automata of the rules. These process models naturally satisfy the prescribed behavioral constraints of the rules. With the domain knowledge encoded in the temporal business rules, an executable service composition program, e.g. a BPEL program, can be further generated from the process models.

1 Introduction

The service-oriented computing paradigm, which is currently highlighted by Web services technologies and standards, provides an effective means of application abstraction, integration and reuse with its loosely-coupled architecture [1]. It prompts the use of self-describing and platform-independent services as the fundamental computational elements to compose cross organizational business processes. Executable Service composition languages including BPEL [2] and BPMN [3] have been created as effective tools for developing applications in this paradigm.

In the process of developing service-oriented applications, it is essential to ensure that the service composition being developed possesses the desired behavioral properties specified in the requirements. Unexpected application behaviors may not only lead to mission failure, but also may bring negative impact on all the participants of this process.

One of the typical solutions to this problem is through verification: by formally specifying the behavioral properties and then applying the model checking technique to ensure the conformance of the application to these properties. A bunch of research works have been published on the verification of service compositions in BPEL BPEL [4, 5, 6]. We also have proposed a pattern based specification language PROPOLS and used it on verifying BPEL programs [7]. One of the significant features of our approach is that PROPOLS is an intuitive, software practitioner accessible language that can be used by business experts to express temporal business rules.

Synthesis is the process of generating one specification from another at an appropriate level of abstraction, while some properties of the source specification are kept in the target one. Comparing with verification, the synthesis approach gives further benefits to the developers: Except for ensuring the property conformance, part of the application design and programming work is done automatically.

In this paper, we propose a synthesis framework and associated techniques to generate service composition process models from a set of PROPOLS temporal business rules. The PROPOLS rules prescribe the occurrences or sequence patterns of business activities in a business domain. The behavioral model of a set of rules can be achieved by translating each rule in the set into a semantics-equivalent Finite State Automaton and then composing them into another FSA with the logical operators defined upon FSA. A set of process models of the targeting service composition can be synthesized by analyzing the acyclic acceptable paths of the resulting FSA. Dedicated techniques include path-finding, branch structure introduction, and parallel structure introduction. Because of the "looseness" of the temporal business rules specification, which means the specification is incomplete, some of the generated process models are either trivial or meaningless to the developer. At this time, the developer can introduce new pertinent business rules to get a more precise result of the process models. When a satisfactory process model is generated, it can be further transformed into a BPEL program by discovering reusable Web services based on the ontology information encoded in the business activities. In a word, our synthesis framework offers an "intuitive specification" and then "correct by auto-construction" solution bring benefits to either a novice or an expert software developers.

The rest of the paper is organized as follows. Section 2 presents an overview of our synthesis framework. Section 3 explains the synthesis process and techniques in detail by an example from the e-business domain. Section 4 discusses the related work and we conclude the paper in Section 5.

2 Overview of the Synthesis Framework

Fig. 1 summarizes the main components of our synthesis framework. The shadowed ovals indicate the three major phases, specification, synthesis, and transformation, where iterations between specification and synthesis are usually necessary to get a precise process model.

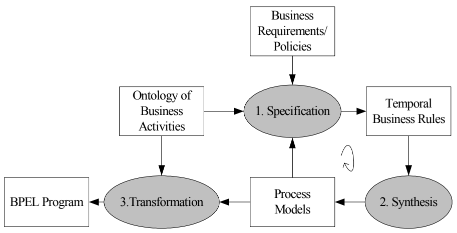


Fig. 1. Overview of the Synthesis Framework.

The focus of this paper is on the synthesis phase.

Specification

Temporal business rules state the occurrence or sequencing orders between business activities prescribed by some business requirements or policies. Business activities represent reusable services in a business domain, either coarse-grain services exposed beyond organization boundary or fine-grain services extracted from function libraries. A taxonomy or ontology can be used to organize business activities for effective browsing and searching.

We use PROPOLS [7] to specify temporal business rules. PROPOLS is a high-level temporal constraints specification language. The main constructs of PROPOLS are property patterns [8, 9] abstracted from frequently used temporal logic formulas. A logical composition mechanism allows the combination of patterns to express complex requirements. Below we briefly describe the key constructs and semantics of PROPOLS.

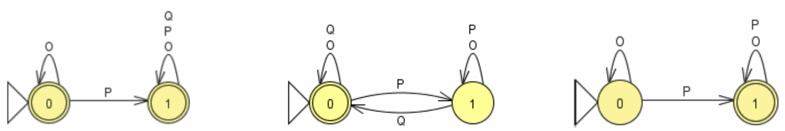
PROPOLS has two main constructs: basic patterns and composite patterns. Fig. 2 shows the form of basic patterns. The constructs on the left are temporal patterns and those on the right are scopes. A temporal pattern specifies what must occur and a scope specifies when the pattern must hold. The P, Q, R, and S in the figure denote events parameters (or business activities in this work) and n is a natural number.

Every temporal pattern has the intuitive meaning by its name, e.g. "precedes" means precondition relationship, "leads to" means cause-effect relationship, "p is absent" means p can not occur, "p is universal" means only p can occur, and "exists" defines the occurrence time of an event. Scope "globally" refers to the whole execution period of an application. Scope "before S" refers to the portion before the first occurrence of S, and so on.

$$\begin{array}{lcl}
\text{Temporal Patterns} & & \text{Scopes} \\
\hline
\left.\begin{array}{l}
P\ \textbf{precedes}\ Q \\
P\ \textbf{leads to}\ Q \\
P\ \text{is}\ \textbf{absent} \\
P\ \text{is}\ \textbf{universal} \\
P\ \textbf{exists}[\begin{bmatrix}\text{at most} \\ \text{at least}\end{bmatrix}\ n\ \text{times}]
\end{array}\right\} & \times & \left\{\begin{array}{l}
\text{globally} \\
\text{before}\ S \\
\text{after}\ R \\
\text{between}\ R\ \text{and}\ S \\
\text{after}\ R\ \text{until}\ S
\end{array}\right.
\end{array}$$

Fig. 2. Basic Patterns.

Every basic pattern has a one-to-one relationship with a predefined FSA which precisely expresses its semantics. E.g. Fig. 3 illustrated the corresponding FSA of basic patterns "precedes", "leads to", and "exists" with "globally" scope, where the symbol O denotes any other events than the named events. Fig. 3(a) says that before P occurs, an occurrence of Q is not accepted. Fig. 3(b) says that if P has occurred, an occurrence of Q is necessary to drive the FSA to a final state. And Fig. 3(c) says that only the occurrence of P can make the FSA reach a final state.



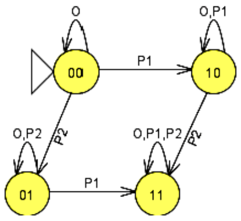
(a) P **precedes** Q **globally** (b) P **leads to** Q **globally** (c) P **exists globally**

Fig. 3. FSA Semantics of Basic Patterns.

Composite patterns are constructed by the logical composition of basic patterns. The syntax of composite patterns in BNF is:

Pattern = basic pattern | composite pattern
Composite pattern = not Pattern | Pattern and Pattern | Pattern or Pattern | Pattern xor Pattern

The semantics of composite patterns can be expressed by the logical composition defined upon FSA [10]. E.g. Fig 4 describes the logical composition between two basic patterns: "P1 exists globally" and "P2 exists globally". The states are the Cartesian production of the two FSA and the final states are determined by the logical operator used.



Logic Operator	Final States
And	11
Or	01,10,11
Xor	01,10
Imply	00,01,11

Fig. 4. FSA Semantics of Basic Patterns.

Synthesis

Fig. 5 describes the steps of synthesis.

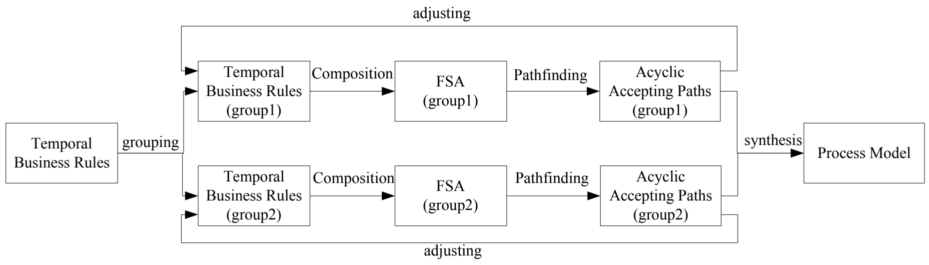


Fig. 5. Synthesis Process.

The main purpose of grouping is to separate concerns. One grouping strategy is by the goals/sub-goals of the business activities involved. E.g. if a set of business activities is classified under "ProcessOrder" goal, then all the temporal rules defined upon these business activities are in one group. Grouping can reduce the number of temporal rules that should be considered at a time, which reduces the complexity of the total synthesis process.

Based on a group of temporal rules, we get the corresponding semantic-equivalent FSA of each rule and then compose them into one FSA based on the logical composition operators defined upon FSA [10]. Usually, the "and" operator is used because we want all the rules be satisfied, in this situation, the resulting FSA precisely describe the all-satisfying semantics of this group of rules. Every string in the accepting language of the resulting FSA is a justified execution path of the related business activities, which conforms to all the rules in the group (Yu et al., 2006b).

In fact Many accepting paths are infinite because of the loop in the resulting FSA. We just find all the acyclic accepting paths (AAPs in short), because a loop can't introduce new final state to the path.

Every AAP is a sequence of business activities satisfying the group of rules. If the generated AAPs can't satisfy the user's expectation, e.g. the number is too big or only contains trivial solutions, the user can refine his requirements by introducing additional temporal rules between business activities to get more precise AAPs.

The last step is to synthesis all the generated AAPs into a process model. First the user should pick one AAP from each group and connect them manually. Then techniques that automatically introduce branch and parallel structures will be used to generate a process model. Temporal rules defined between groups also will be used to check the validity of the process model.

Transformation

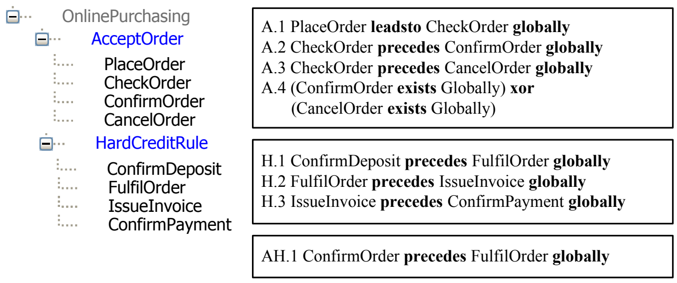
The resulting process model is transformed into the control flow constructs, e.g. "sequence", "switch", and "flow", in BPEL. The ontology of business activities will be used to discover reusable Web services and transformed into the "invoke" action in BPEL.

3 The Synthesis Process

In this section we describe our synthesis method and techniques in detail by an example. This example is adapted from a frequently appeared online purchasing scenario in the e-business domain. Our scenario accepts online orders and then processes them by the "Hard-Credit" business rule. To accept an order, this order must be checked for validity, the customer who places the order will receive either a confirmation or cancellation of the order based on the checking result. The purpose of the "Hard-Credit" rule is to protect the benefits of both the customer and the business provider. This rule states that the customer MUST pay when the order is fulfilled, and the payment is made only after the customer has received the goods and invoice. A third-party trustee, e.g. a bank, is necessary to implement this rule, first the customer deposit the payment to the bank, then the payment is transferred to the provider if the customer received the desired goods.

Specification

Fig. 6 shows the business activities solicited from the above-stated scenario and the temporal business rules defined upon them. Business activities and temporal rules are classified by two sub-goals: “AcceptOrder” and “HardCreditRule”. Note that there is also one temporal rule, AH.1, defined between groups.



A.1 PlaceOrder **leadsto** CheckOrder **globally**
A.2 CheckOrder **precedes** ConfirmOrder **globally**
A.3 CheckOrder **precedes** CancelOrder **globally**
A.4 (ConfirmOrder **exists** Globally) **xor** (CancelOrder **exists** Globally)

H.1 ConfirmDeposit **precedes** FulfilOrder **globally**
H.2 FulfilOrder **precedes** IssueInvoice **globally**
H.3 IssueInvoice **precedes** ConfirmPayment **globally**

AH.1 ConfirmOrder **precedes** FulfilOrder **globally**

Fig. 6. Business Activities of the Online Purchase Example.

Fig. 7 shows the FSA generated by the and-composition of A.1~A.4 using the verification tool introduced in [7]. Every path from the initial state (0) to the final states (12 and 15) is a valid run that satisfies rule A.1 to rule A.4.

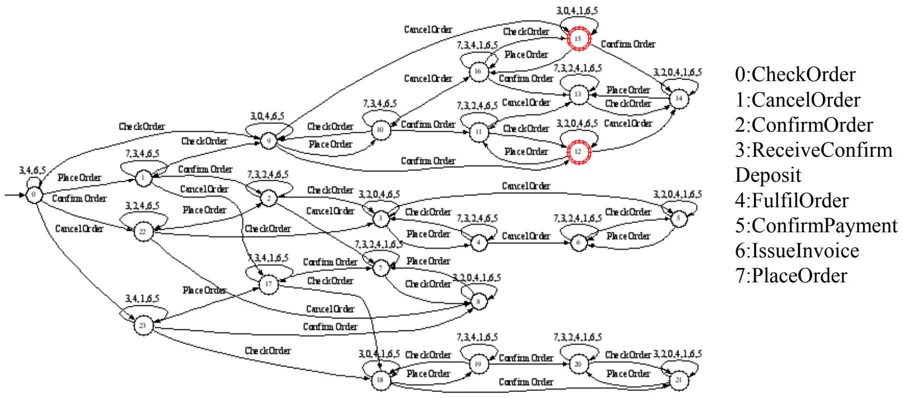


Fig. 7. FSA Composed from A.1~A.4.

Path-Finding

The algorithm of finding all the acyclic paths in a FSA is described in Fig. 8. This is a variation of the Depth-First-Search algorithm [11]. The most significant modification is that a state can be visited N times if it has N non-loop input edges (is on N different non-loop paths starting with the initial state). For example, state 9 in Fig. 7 has 2 non-loop input edges, so it will be visited 2 times when searching.

```
Global var: int counter, int[] order;
Procedure fsaAcyclicPath(FSA G) {
   counter = 0;
   order = new int[G.numberOfStates];
   for (int t = 0; t < G.numberOfStates; t++){
      order[t] = NotVisited;}
//search all the paths start with the initial state
```

```
        searchC(0);
   }
   Procedure searchC(int v) {
      order[v] = Visited;
      AdjacentList A = G.getAdjacentList(v);
      for (Node t = A.begin(); !A.end(); t = A.next()){
          if (order[t.v] == NotVisited){
              addEdge2Tree(v,t);
              searchC(t.v);
   //mark the state as not-visited when move to a new path
              order[t.v] = NotVisited;}}}
```

Fig. 8. Algorithm for FSA Acyclic Path-Finding.

Using the above path-finding algorithm to the FSA in Fig. 7, we can get all the acyclic paths starting from the initial state. Fig. 9 is an excerpt of the path-tree where the concentric circles are the final states.

From the generated path-tree, we can totally get 8 AAPs: 1.(Place, Check, Cancel)[1] , 2.(Place, Check, Confirm), 3.(Place, Check, Place, Cancel, Check), 4.(Place, Check, Place, Confirm, Check), 5.(Check, Place, Cancel, Check) , 6.(Check, Place, Confirm, Check) , 7.(Check, Cancel), 8.(Check, Confirm). Clearly not every AAP fits the user's need. At this time, the user can add rules to get more precise AAPs. E.g. if one extra rule, "A.5 PlaceOrder precedes CheckOrder globally", is introduced, the number of the above AAPs will be reduced to 4, only 1~4 are left.

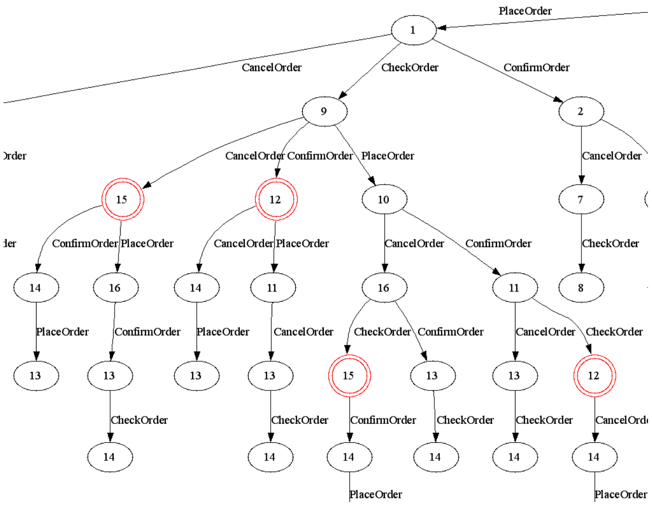


Fig. 9. Excerpt of the Path-Tree.

Synthesis

After all the satisfying AAPs are generated, the user can pick one AAP from each group and connect them manually to build the initial process model which only con-

1 "PlaceOrder" is shorten as "Place" if no ambiguity is introduced. The same rule applied to other business activities.

tains sequence structures. Temporal rules defined between groups will be used to check the validity of such connections.

A heuristic method is used to introduce branch structures into the initial process model: If a rule has the form like “P exists globally xor Q exist globally”, e.g. rule A.4, we introduce a branch between P and Q. The justification of this method is that the process model with the branch will be verified correct against the temporal rule.

The introduction of parallel structures is based on interleaving assumption, which states that two events are concurrent if their occurring order does not change the consequence (Milner, 1989). Based on this assumption, if we have two business activities P and Q, P next to Q in one AAP and Q next to P in another AAP, which means the occurring order of P and Q has nothing to do with the execution consequence, we are sure that P and Q can be put in a parallel structure. E.g. if we compose all the rules in Fig. 6, we can find that “ConfirmOrder” and ConfirmDeposit” can go in parallel.

Using the above-stated methods and techniques, a possible synthesized process model is shown in Fig. 10.

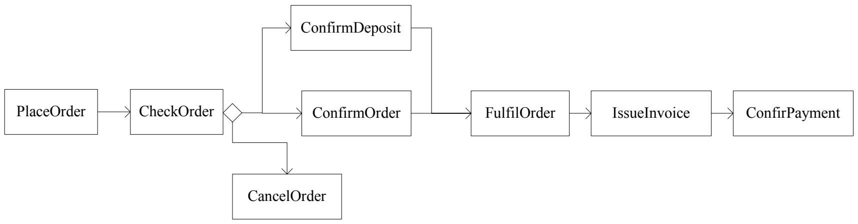


Fig. 10. A Possible Synthesized Process Model.

4 Related Work

A body of work has been reported on generating process models in the area of service oriented computing. Berardi et. al. use situation calculus to model the actions of Web services, and generate a tree of execution paths [13], they also use FSA to model the actions of individual services and then synthesis the service composition FSA [14]. In [15], Wu et. al. discuss how to synthesis Web service compositions based on DAML-S using an AI planning system SHOP2. Duan et. al. synthesis a BPEL abstract process from the precondition and post-condition of individual tasks [16]. Most of the above works are based on AI planning. One problem with AI planning synthesis is that planning focuses on generating a sequential path for conjunctive goals and does not consider generating process constructs like conditional branching and parallel execution.

More generally, there are also some approaches on generating formal behavioral models from another formalism. E.g. Beeck et. al. use Semantic Linear-time Temporal Logic to synthesis state charts [17]. Uchitel et. al. use Message Sequence Charts to synthesis Finite Sequential Processes[18]. The most significant difference between our approach and theirs is that our process model is more close to the final program, while their models are more abstract and suitable for reasoning the general properties of the system.

5 Conclusion

In this paper, we have presented a framework and associated techniques to semi-automatically synthesis service composition process models from temporal business rules. This framework is supposed to give much help to common software practitioners, the rule specification language PROPOLS is intuitive and works at the business level, a "correct" process model can be generated semi-automatically, which facilitates daily programming work and finally brings benefits to both the novice and the expert software developers.

Currently, we are working on the transformation phase of the framework. In the future, we plan to integrate this framework with some graphical service composition editors, e.g. *ActiveBPEL Designer* [19].

References

1. Alonso, G., Casati, F., Grigori, Kuno H., Machiraju, V.: Web Services Concepts, Architectures and Applications. Springer-Verlag (2004).
2. Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0 Workgraft. http://www.oasisopen.org/committees/download.php/10347/wsbpelspecification-draft-120204.htm (2004)
3. BPMI: Business Process Modeling Language. http://www.bpmi.org/ (2002).
4. Foster, H.: A Rigorous Approach to Engineering Web Services Compositions. PhD thesis, Imperial College London. http://www.doc.ict.ac.uk/~hf1 (2006).
5. Stahl C.: A Petri Net Semantics for BPEL. Informatik-Berichte 188, Humboldt-Universitat zu Berlin, June 2005 (2005).
6. Fu, X., Bultan T., Su J.: Analysis of Interacting BPEL Web Services. In Proc. 13th World Wide Web Conf. New York, NY, USA (2004) 621-630.
7. Yu, J., Phan, T., Han, J., Jin, Y., et al: Pattern Based Property Specification and Verification for Service Composition. In Proc. 7th Int. Conf. on Web Information Systems Engineering. Springer-Verlag, LNCS 4255. Wuhan, China (2006) 156-168.
8. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite state Verification. In Proc. 21th Int. Conf. on Software Engineering. Los Angeles, CA, USA (1999) 411-420.
9. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: A System of Specification Patterns. http://www.cis.ksu.edu/santos/spec-patterns (1997).
10. Yu, J., Phan, T., Han, J., Jin, Y.: Pattern based Property Specification and Verification for Service Composition. Technical Report SUT.CeCSES-TR010. CeCSES, Swinburne University of Technology, http://www.it.swin.edu.au/centres/cecses/trs.htm (2006).
11. Sedgewick, R.: Algorithms in Java, Thrid Edition, Part 5: Graph Algorithms. Addison Wesley (2003).
12. Milner, R.: Communication and Concurrency. Prentice-Hall (1989).

13. Berardi, D., Calvanese, D., Giuseppe, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In Proc. 1st Int. Conf. on Service Oriented Computing. Trento, Italy (2003).
14. Berardi, D., Glancomo, G., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In Proc. 2st Int. Conf. on Service Oriented Computing. . New York, USA (2004).
15. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.: Automating DAML-S web services composition using SHOP2. In Proc. 2nd Int. Semantic Web Conf. Florida (2003).
16. Duan, Z., Bernstein, A., Lewis, P., Lu, S.: A Model for Abstract Process Specification, Verification and Composition. In proc. of the 2nd Int. Conf. on Service Oriented Computing. New York, USA (2004).
17. Beeck, M., Margaria, T., Steffen, B.: A Formal Requirements Engineering Method for Specification, Synthesis, and Verification. In Proc. 8th Int. Conf. on Software Engineering Environment. Washington, DC, USA (1997).
18. Uchitel, S., Kramer, J., Magee, J.: Synthesis of Behavioral Models from Scenarios. IEEE Trans. On Software Engineering Vol.29 2 (2003) 99-115.
19. ActiveBPEL Designer. http://www.activenedpoints.com/products/activebpeldes/ (2007).

Author Index

Proceedings of the
1st International Workshop on
Architectures, Concepts and Technologies
for Service Oriented Computing - ACT4SOC 2007
ISBN: 978-989-8111-08-1
http://www.icsoft.org