RC23821 (W0512-008) December 1, 2005
Computer Science

IBM Research Report

Proceedings of the First International Workshop on Engineering Service Compositions (WESC'05)

Amsterdam, The Netherlands, December 2005

[1]Christian Zirpins, [2]Guadalupe Ortiz
[1]Winfried Lamersdorf, [3]Wolfgang Emmerich (Eds.)

[1]University of Hamburg, Germany
[2]University of Extremadura, Spain
[3]University College, London, UK

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

Preface

The emerging paradigm of service-oriented computing (SOC) introduces groundbreaking concepts for distributed- and e-business processing. In fact, they are believed by many to radically change the way software applications are designed, architected, delivered and consumed. Web Services, which constitute the heart of SOC, are autonomous platform-independent computational elements that can be described, published, discovered and accessed over the Web using standard protocols. Service-oriented architectures (SOA) leverage the foundational capabilities of computational service models. They provide a technological as well as conceptual framework for new classes of cooperative business applications: agile networks of collaborating business applications distributed within and across organisational boundaries.

Consequently, SOA not only includes software technologies to aggregate atomic services into composite services (a.k.a. service composition). It also comprises the software engineering methodology to turn composite services into cooperative business applications (a.k.a. service engineering). These aspects are highly interrelated. Service composition became one of the major software technology approaches for composing multiple coarse-grained applications over the Web. Thereby, it originated innovative composition models with pioneering concepts for e.g. generation, coordination and aggregation of compositions. Such models introduce significantly different ways of managing business connectivity. Thereby they make a strong impact on application semantics and vice versa. Thus, software engineers have to take into account the impacts of different service composition models. This is crucial for them to guarantee a certain quality level of SOA-based cooperative business applications with respect to functional and non-functional requirements.

The upcoming discipline of service engineering generally benefits from former research on component- and aspect-oriented software engineering methodologies. Meanwhile, there are already promising results on novel conceptual and technological tools to support the development processes of cooperative business applications. However, such tools need to be increasingly aligned with service composition technology. Joint approaches on engineering service compositions face several open problems and challenges. On the technology side there is still neither an agreement on service composition models and languages nor on their scope of application; let alone experiences on mission critical operation. As regards methodology, reference architectures of service-oriented cooperative information systems taking into account particularities of the service composition lifecycles are just at the beginning. This is just to name a few issues.

Accordingly, this workshop brought together experts from service composition technology and service engineering methodology; researchers and practitioners from industry and academia. It fostered discussions about problems and challenges that particularly arise during the practical combination of both fields

of expertise for the realisation of service-based, cooperative business information systems. In reply to the call for paper a brought variety of submissions illustrated the relevance of the topic. All received contributions where carefully reviewed by the international program committee. In the end, 13 papers were selected with focus on quality, originality and topical relatedness. This set covers perspectives and concepts in the form of ideas as well as results on a wide range of topics related to engineering software by service composition.

Finally, we would like to thank the people without whom this workshop would not have been possible: the members of the program committee as well as the assisting reviewers for their tremendous work, the organisers of the ICSOC conference and especially Frank Leymann and Mike Papazoglou for their help in planning and implementing the workshop, Paco Curbera for his support in publishing the proceedings, George Feuerlicht for his co-operation and of course the many authors for their great contributions.

November 2005

Christian Zirpins
Guadalupe Ortiz
Winfried Lamersdorf
Wolfgang Emmerich

Organisation

Workshop Chairs

Wolfgang Emmerich, University College London, UK
w.emmerich@cs.ucl.ac.uk

Winfried Lamersdorf, University of Hamburg, Germany
lamersdorf@informatik.uni-hamburg.de

Guadalupe Ortiz, University of Extremadura, Spain
gobellot@unex.es

Christian Zirpins, University of Hamburg, Germany
zirpins@informatik.uni-hamburg.de

Program Commitee

M. Aiello, University of Trento, Italy
F. Casati, HP Palo Alto, USA
V. D'andrea, University of Trento, Italy
S. Dustdar, Technical University of Vienna, Austria
W. Emmerich, University College London, UK
G. Feuerlicht, Technical University of Sydney, Australia
H. Foster, Imperial College London
M. Hauswirth, EPFL Lausanne, Switzerland
J. Hernández, University of Extremadura, Spain
W. Lamersdorf, University of Hamburg, Germany
M. Mecella, University of Rome, Italy
G. Ortiz, University of Extremadura, Spain
M. Papazoglou, Tilburg University, The Netherlands
P. Plebani, Politecnico di Milano, Italy
T. Risse, Fraunhofer Society, Germany
C. Roland, University of Paris, France
S. Tai, IBM Watson, USA
M. Weske, HPI at University of Potsdam, Germany
A. Wolf, University of Boulder, USA
J. Yang, Macuquarie University, Australia
C. Zirpins, University of Hamburg, Germany

Additional Referees

L. Braubach
G. Frankova
G. Gr
M. Husemann
N. Kokash
A. Lazovik

H. Meyer
A. Pokahr
F. Rosenberg
B. Schmit
H. Schuschel

Table of Contents

An Organisational Approach to Building Adaptive Service-oriented Systems

Alan Colman and Jun Han

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Victoria, Australia
{acolman,jhan}@swin.edu.au

Abstract. The relationships between the loosely coupled services of a composition can be non-deterministic and unreliable. This requires that a composite application have the ability to adaptively reconfigure itself to continuously satisfy the system requirements. Imperative composition approaches, such as BPEL, do not provide the abstractions necessary to create adaptive compositions. In this paper we introduce an organisation-oriented, application-centric service composition framework that aims to achieve adaptive application behaviour from non-deterministic and unreliable services. The responsibility for coordinating and managing service interactions resides with a coordination/management layer of the application itself. This layer involves two types of essential elements: service contracts and service organisers. The service contracts define and regulate the relationships between the services in a composition. The service organisers control the bindings between service types and individual services, and can create and revoke contracts between services within the composition in order to adaptively reconfigure the application in response to changing conditions. An adaptive service composition can be formed by connecting and configuring services through dynamically formulated contracts, under the control and management of service organisers.

1 Introduction

In order to compose Web services with other services or applications, the services need to be functionally compatible, the interactions between the services need to be well-ordered, and the composed behaviour must meet any requirements for the system as a whole. However, in the open and distributed environment of the Internet, the relationship between loosely coupled services can be non-deterministic. In an application-centred architecture, the central application needs to be able to meet its functional and non-functional requirements, even if the Web services it relies upon are to some degree unpredictable. In particular, if an application needs to guarantee a certain level of performance, and yet relies on distributed Web services to provide some functionality, the application needs a mechanism to manage the quality of service (QoS) of its subsidiary services. For example, if the application is a work flow

scheduling system, the system needs to respond dynamically to variations in the performance of its constituent services. It must monitor and restructure the interactions of the services as demand changes and service loads change. As such, it requires the composite system to have *management* capabilities.

Much work has been done on the preparation of standards to allow the development of Web service compositions that are well-behaved and meet the requirements of the composite system. While the point-to-point simple interactions have reached a level of maturity in their standardisation and implementation, achieving reliable behaviour from more complex configurations of Web services is still an open problem. The development of standards to address such behavioural and non-functional aspects of complex composed services is being undertaken on a number of fronts: for example, coordination [3], choreography [11], orchestration [4], and management [10].

Many of these standardisation efforts define a management or coordination level of abstraction. In these standards management and coordination functions are encapsulated as services with a well-defined Web service interface. No clear guidance is provided on how the programmer can use these various overlapping and sometimes incompatible middleware standards to achieve a well regulated, adaptive application. What is needed is a framework that provides the necessary management and coordination functions that the application programmer can readily extend to suit domain and application specific requirements. In this paper, we propose such a framework – the Role Oriented Adaptive Design (ROAD) framework for Web services. Like a number of WS standards, the ROAD framework defines a separate coordination/management layer. However, this layer is superimposed on the pre-existing decoupled application entities and creates an organisational structure for it, rather than being a separate encapsulated service provided by the middleware. The primary responsibility for coordination and management of the interactions resides with this coordination/management layer of the application.

The structure of the paper is as follows. Section 2 defines the concept of an adaptive system in terms of managed indirection of *instantiation* and *composition*. Section 3 outlines a role-based architectural framework that supports managed indirection to achieve adaptivity. Section 4 concludes and briefly discusses of related work.

2 Adaptive systems – the management of indirection

We define an adaptive system as one that can reconfigure itself in response to changes in its environment, or in response to changes in its own goals or capabilities. The perturbation between a system and its environment can include the availability and variable performance of Web services that it relies on to function. Even if a system's services have unpredictable performance, we still need to achieve some acceptable level of system-wide performance. A system needs to be able to recognise when one of its services is underperforming, and then take remedial action: action such as reorganising the work load, reconfiguring the interactions between services, or replacing the underperforming service. Adaptable systems are those that can be

readily recomposed. To achieve recomposition, the system structure must be flexible, with indirection between the entities in the system.

Indirection in Web service compositions can be created in two dimensions — *indirection of instantiation*, and *indirection of composition*. Firstly, abstract description of services can be distinguished from their concrete implementation (as in the separation of the abstract and concrete parts of a WSDL specification [12]). In an adaptive system, such binding should be able to be created and destroyed at runtime. Such dynamic binding is well supported in Web services (in specification at least) through service discovery and selection mechanisms.

The second type of indirection is found in the association between the abstract services. Associations between service types might be hard-coded references in the implementation of a client service, or hard-coded using an imperative composition language such as BPEL [4]. Such associations are abstracted by using reference variables, or by explicitly representing an association type (such as a contract or partner-link type as in BPEL). However, in the absence of other mechanisms, all possible runtime *determinations* (resolutions) of the association indirection must be anticipated at design time (e.g. in BPEL as cases in a switch structure). The indirection in the association is not *per se* adaptive because the associations are hard-coded when the service or the composition script is implemented.

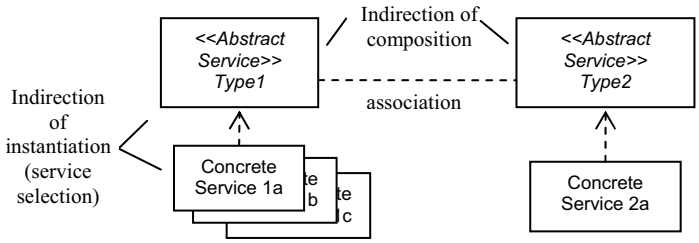

Fig. 1 Two dimensions of indirection in Web services.

Fig. 1 illustrates the two types of indirection that give software systems the flexibility needed to be adaptive: namely indirection of *instantiation* and *composition*. While the mechanism of runtime determination of instantiation is supported in Web services, the determination of composition between service types is often fixed during implementation.

In a functioning system any indirection must be determined before or during runtime. An adaptive system needs to manage its indirection: that is, it needs to be able to dynamically create bindings between its loosely coupled elements on both the above dimensions in response to changing demands and changing environment. To do this, the adaptive system needs to be able to perform the following management and binding functions:

- monitor the performance (or other quality attributes) of its current configuration;
- have access to alternative services (with various performance/quality characteristics) and the ability to select between services
- ensure associated services are functionally and interactionally compatible
- check the validity and evaluate the performance of alternative compositions
- reconfigure its bindings and associations.

The basic Web service standards of WSDL and UDDI partially address some of these functions (basic functional compatibility and service selection respectively). There has been a recent proliferation of standards that address some of the other functions listed above [3,8,10,11]. These standards are to an extent overlapping and incompatible, and they do not provide clear guidance on how to integrate and use such standards, if we want to develop an adaptive application as characterised above. For example, BPEL allows us to define compositions from services that are compatible in signature, but does not support performance monitoring in its orchestration model other than fault handling and compensation. If the programmer wants to monitor the performance of a service, the monitoring code is tangled with the orchestration script. As well as providing the necessary indirection, an adaptive architecture must show what elements in the system are responsible for the binding and management of that indirection, and how this can achieve adaptive adjustment.

3 An Adaptive Framework for using Web Services

In this section we introduce a framework to facilitate the development of applications built from Web services. We call this framework Role Oriented Adaptive Design (ROAD). The design aims of this framework are as follows:

- facilitate the development of adaptive applications that can perform reliably while using Web services whose performance is unreliable
- support the managed indirection of instantiation and composition
- provide a framework of constructs that can be extended by the programmer to create coordination and management for adaptive composites
- allow the management/coordination layer to be coded as a separate concern from the functional roles and services
- have compositions that can be imposed on services without the services needing to be (necessarily) written to horizontal standards such as WS-Coordination.

This section outlines the basic elements of the ROAD framework, and shows how it can be used to create adaptive compositions as defined above. The adaptive framework for using Web services introduced here is based on the conceptual separation of *roles* from the entities that play those roles, and the association of such roles with contracts. Applications are viewed as organisations — *goal-driven* networks of roles bound together by contracts. Roles can be played by various *players*, in much the same way as a role in a business structure may be played by various employees, or outsourced to external organisations. Similarly, roles in the adaptive service-oriented application can be played by Web services or other components that are within the organisation, or by external Web services outside the application's immediate scope of control. Players can be dynamically bound/unbound (indirection of instantiation) to roles as demands on the application changes, or as the players' performance varies.

Contracts associate organisational roles. They monitor and regulate interactions between the roles. As all roles (as opposed to the players) are internal to the organisation, contracts are also internal (although these internal contracts may be mirrored in external SLAs). Contracts define the mutual obligations of the participant roles in an organisational context. They define what interactions are permissible or

required by the participant roles, and can be used to enforce sequences of interactions (conversations). Contracts can also be used to set performance or other quality conditions on the roles' interactions, and monitor those interactions for compliance to those conditions (performance management). Contracts thus encapsulate both the coordination and the performance management of interactions.

Organisers make and break the bindings between organisational roles and players (service selection), and create and revoke the contracts between the roles. They can thereby create various configurations of roles and players. Organisers set performance requirements for the contracts they control, and receive performance information from those contracts. Organisers are themselves a role-player pair, so that role players of varying capability can be bound to the organiser role. In short, organisers provide the adaptivity to the application by managing the indirection of composition and instantiation. Organisers are responsible for the configuration of a set of roles and contracts. We call such configurations *self-managed composites*. The following subsections explain these concepts in more detail.

3.1 Determining Indirection with Bindings and Contracts

At runtime, services are bound to roles, and roles are associated with each other using contracts. Functional roles are application entities that hold abstract service definitions. Because the services that play roles can be transitory (for example, there may be no service currently available to play a role), roles also maintain any state (e.g. message queues) associated with their position in the organisation. To illustrate the concept of a role, consider a highly simplified segment of a production system application for a Widget manufacturing department. Thingies are obtained from an external supplier and made into Widgets by an Assembler. A Foreman supervises, among other things, the work schedule of the Assembler.

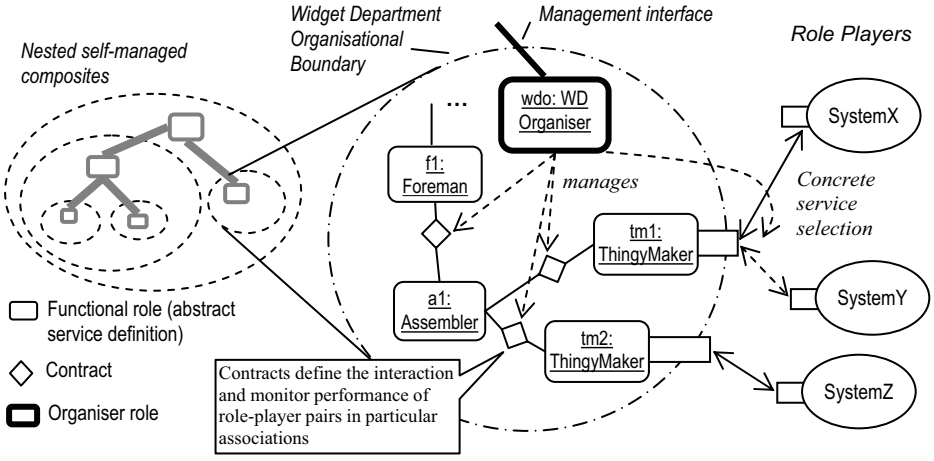

Fig. 2. Creating flexible role-structures with role-player bindings and role-role contracts

Fig. 2 above illustrates the relationship between a role within an organisation and an external service that plays that role. The role can be thought of as a proxy within the

organisation for the external service. The role includes a WDSL abstract service definition that is made concrete with a service endpoint reference when a service is bound to the role. Service selection (eg. between ServiceX and ServiceY) is not the responsibility of the role itself, but the responsibility of the composite's organiser wdo (as discussed below).

Contracts between roles can also be used to create flexibility. By creating and/or revoking contracts, the topology of the composition of roles can be altered. For example, if a role instance tm1 of type ThingyMaker (played by ServiceX) is unable to meet the requirements of an Assembler a1. An alternative to replacing ServiceX with a more capable service (say ServiceY) would be to add another role instance tm2 of type ThingyMaker (played by ServiceZ) to the role structure. This changes the topology of the organisation. Contracts, however, provide more than simple references between role types. They monitor and regulate interactions between the roles. Contracts define the mutual obligations of the participant roles. They define what interactions are permissible or required by the participant roles, and can be used to enforce sequences of interactions. Contracts can also be used to set performance and quality conditions on the interactions, and monitor those interactions for compliance to those conditions (performance management). Contracts thus encapsulate both coordination and performance management of interactions. A more detailed discussion of ROAD contracts, their contents and how they are implemented can be found in [6,7].

Functional contracts specify the required performance of interactions between the parties, and they monitor and store actual performance. They can be thought of as service level agreements (SLAs) for the roles in the organisation. The monitoring of service performance using the ROAD framework is extrinsic; that is, it is performance of a concrete role-player pair *as measured from* the application that is using the service. The application does not have to rely on abstract descriptions of service quality (although this may be helpful as a starting point for service discovery) because it can measure the time elapsed (or other change of state) between, for example, request and response messages. Basic time and count based metrics (similar to those defined in WSDM-MoWS [10]) are provided as part of the ROAD framework and do not have to be coded by the application programmer. However, the programmer can extend the performance measurement capabilities with domain-specific metrics (for example cost or reliability functions) if required. There is another advantage to storing performance measurements in the contract rather than in, say, a log of *all* interactions of a service. The advantage of contract-centric monitoring is that the performance profile of a service relates to a particular client. This profile may vary depending on the client identity and location of the service; for example the client's priority or the capacity of the connection between the client and the service.

3.2 Internal Contracts and External SLAs.

The above discussion only deals with 'internal' contracts between roles within the organisational boundary, although those roles may be proxies that represent external services. Some interactions, particular long-lived ones, may need an external contract between the role (Web service proxy) and player (concrete Web service). The ROAD framework does not preclude the use of external coordination mechanisms such as WS-Coordination [3], or external contracts such as WS-Agreement [8]. Indeed, the

internal contract that binds the proxy role to the rest of the system can serve as a template for defining external coordination contexts or contracts.

The concept of a ROAD contract also maps naturally to Service Level Agreements at the business level, in part because roles (such as ThingyMaker) map better to business entities than do, say, BPEL activities. Discussion of these mappings is beyond the scope of this paper, other than to note that the definition of an internal contract provides the desired requirements (both protocol and performance) from the organisation's point of view. What can be negotiated in an external contract also provides constraints for what can be defined in the associated internal contract.

3.3 Organisers and Self-managed Composites

A self-managed composite contains functional roles bound by contracts and an *organiser* role. From the enclosing system's perspective, a self-managed composite is a role player with a management interface. Self-managed composites contain roles which may themselves be self-managed composites. This recursive structure enables the nesting of self-managed composites (as in Fig. 2). Self-managed composites have two interfaces – a *functional* interface and a *management* interface. The *functional* interface of a composite is the aggregation of all the Web services it exposes. The *management* interface of the composite is the interface of the *organiser* role. This is a conceptually similar arrangement to an 'out-of-band' manageability interface in MoWS [10] which could be used for this purpose.

The organiser role is responsible for monitoring the performance of the composite's contracts either through notification from its contracts or by polling them. It also reconfigures the composite if environmental perturbation, or a change of requirements, leads to the composite not meeting (or potentially not meeting) its contractual performance obligations. The organiser can achieve reconfiguration by assigning and revoking contracts, and by changing the binding between functional roles and services as shown in Fig. 2. For an organiser to be able to respond adaptively to changing situations, it needs to have at its disposal a range of role/role-players of various performance characteristics, and a mechanism for service discovery such as UDDI. The function of organisers and the dynamics of reconfiguration is not further described here due to space limitations, but a description of adaptive behaviour in self-managed composites, can be found in [5].

4 Related Work and Conclusion

The ROAD adaptive framework for using Web services can be used by an application programmer to create applications that respond to changes in performance or other non-functional requirements. The application can also adapt to performance variations in the Web services it uses. In ROAD, contracts are used to monitor performance/reliability, and enforce protocols. Organisers provide adaptive behaviour to the application by creating and revoking contracts between roles, and by binding roles to services. Organisers control a cluster of roles, contracts and service bindings called a self-managed composite. These composites form a recursive hierarchy that localises organisation but allows system level requirements to be fulfilled.

Like BPEL, our framework is a means of creating executable compositions, but ROAD has roles as its fundamental unit rather than activities. The use of contracts in our framework maps more naturally to inter-organisational business contracts than do process-based approaches. While BPEL is not in itself adaptive, there has been much recent focus on making service composition more flexible. A recent overview of dynamic workflow-based composition can be found in [13]. These approaches focus on adaptive processes using process abstraction, rather than an adaptive role structure as presented here. Monitoring of services has also been addressed in [2] using external 'Smart Monitors' services rather than application-based monitors. However, there is no mechanism for 'monitoring the monitors' (organisers and contracts) as in the recursive ROAD structure. The Cremona framework [9] based on WS-Agreement addresses many of the same issues as the ROAD framework, but focuses on external contracts. These contracts do not control interaction as the management level of a ROAD contract does.

While we have not discussed the implementation of the ROAD framework in this paper, [7] gives a description of an aspect-oriented implementation of ROAD contracts. A similar aspect-oriented approach can be found in [1], where policies are woven into the Web service stubs rather than used to create instances of contracts.

References

[1] Baligand, F. and Montfort, V., "A concrete solution for web services adaptability using policies and aspects," *Proc. of the 2nd Inter. Conf. on Service Oriented Computing,* 2004.

[2] Baresi, L., Ghezzi, C., and Guinea, S., "Smart Monitors for Composed Services ," *Proc. of the 2nd Inter. Conf. on Service Oriented Comp (ICSOC'04),* 2004.

[3] BEA Systems, IBM, and Microsoft *Web Services Coordination (WS-Coordination)*, http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-Coordination.pdf (2004)

[4] BEA, IBM, Microsoft, SAP, AG, and Siebel Systems *Business Process Execution Language for Web Services (BEPL4WS)*, http://ifr.sap.com/bpel4ws/index.html (2003)

[5] Colman, A. and Han, J., "Coordination Systems in Role-based Adaptive Software," *Proc. of the 7th Inter. Conf. on Coordination Models and Languages,* LNCS 3454, 2005.

[6] Colman, A. and Han, J., "Operational management contracts for adaptive software organisation," *Proc. of the Australian Software Eng. Conf. (ASWEC 2005),* 2005.

[7] Colman, A. and Han, J., "Using Associations Aspects to Implement Organisational Contracts," *Proc. of the 1st Inter. Workshop on Coordination and Organisation,* 2005.

[8] Global Grid Forum *Web Services Agreement Specification (WS-Agreement)*, www.gridforum.org/Meetings/GGF11/ Documents/draft-ggf-graap-agreement.pdf (2004)

[9] Ludwig, H., Dan, A., and Kearney, R., "Cremona: an architecture and library for creation and monitoring of WS-agreements," *Proc. of the 2nd Inter. Conf. on Service Oriented Computing (ICSOC'04),* 2004.

[10] OASIS *Web Services Distributed Management -Management of Web Services 1.0,* wsdm-mows-1.0, http://docs.oasis-open.org/wsdm/2004/12/cd-wsdm-mows-1.0.pdf (2005)

[11] W3C *Web Services Choreography Description Language (WS-CDL)* WD-ws-cdl-10-20041012, http://www.w3.org/TR/ws-cdl-10/ (2004)

[12] W3C *Web Services Description Language (WSDL) Version 2.0 Primer*, WD-wsdl20-primer-20050510 , http://www.w3.org/TR/wsdl20-primer/ (2005)

[13] Zirpins, C., Lamersdorf , W., and Baier, T., "Flexible coordination of service interaction patterns ," *Proc. of the 2nd Inter. Conf. on Service Oriented Comp (ICSOC'04),* 2004.

A Multiagent Web Service Composition Engine

Paul Buhler[1], Dominic Greenwood[2] and Alois Reitbauer[3]

[1] College Of Charleston, Charleston SC, USA
[2] Whitestein Technologies AG, Zuerich, Switzerland
[3] Profactor Research, Steyr, Austria

Abstract. This paper reports on a novel approach to the problem of aggregating Web services into meaningful compositions with the aid of a service composition engine constructed with multiagent system and topic map technologies. The engine utilizes a distributed path determination algorithm and due to the multi-threaded nature of the implementation, multiple compositions requests can be serviced concurrently.

1 Introduction

Participation in the global marketplace is no longer restricted to businesses with large Information Technology (IT) budgets. Small and medium sized enterprises are now able to project their presence due to the global connectivity provided by the Internet and the emergence of widely accepted, standards-based platforms for service-oriented computing. These factors have caused a flattening of the IT landscape, as standards-based computing allows the commodization of software. Technologies and methodologies for automating the development of software via compositional approaches are needed in order to reduce the integration friction amongst dynamic collections of software services.

A recent survey of automated Web service composition techniques [6] identifies two broad categories of composition producing systems, those that use workflow-based composition methods and others based upon Artificial Intelligence (AI) planning algorithms. The approach used by the system described in this paper falls loosely into the AI planning category; however, it is distinguished from typical AI planners in several ways.

AI planning algorithms are typically centralized and perform symbolic reasoning. AI planners often operate with a closed world assumption regarding the collection of services eligible for composition and they assume the availability of global and uniform state. We sought to develop a system that addresses these issues by architecting a service composition engine with a multiagent system foundation. With multiagent systems, it is possible to build systems that are both open and scalable and due to their decentralized nature exhibit resiliency in the face of environmental dynamics and partial system failures. The architecture described in this paper is a continuation of our existing research which explores relationships between agent-based and service-oriented computing [1] [3] [2].

2 Architecture

The architecture of the Service Composition Engine is illustrated in Figure 1, which shows the major components of the engine and their interaction. As can be seen, the two primary components of the system are a JADE Agent Platform and a Topic Map System. The latter is used to store WSDL descriptions of Web services to be composed, whilst the former forms the composition engine itself.

The role of each component in Figure 1 and the interactions between them will be described in the following sections.

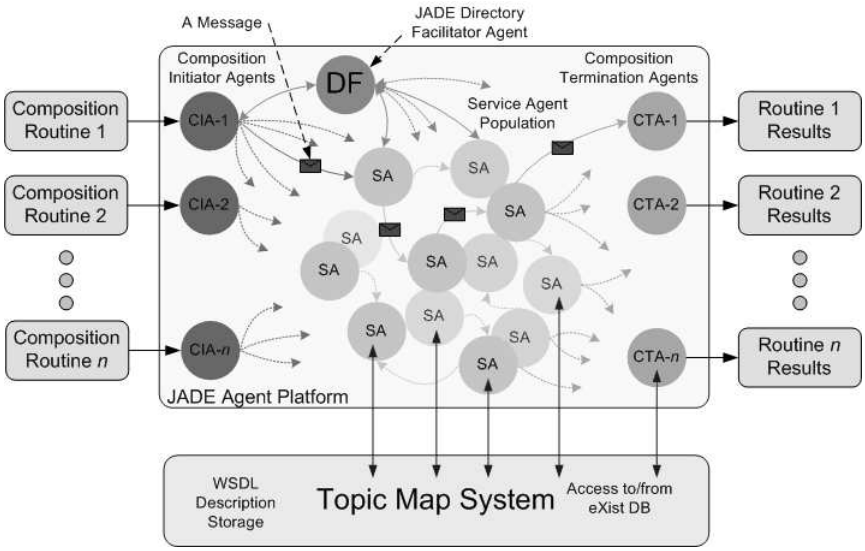

Fig. 1. Service Composition Engine architecture

2.1 Coordination System Architecture

The JADE Agent Platform [7] is the predominant FIPA[4] compliant multiagent system toolkit; it is available open source and supports rapid development of agent-oriented applications. JADE[5] provides both a middleware infrastructure to support agent communication and lifecycle, and the means to build agents consisting of behaviours that cooperatively multitask.

The coordination environment of the SCE is built using the JADE execution model with agents housed within container constructs and communicating

[4] FIPA is a set of public standards for agent-based systems engineering, see http://www.fipa.org/

[5] http://jade.tilab.com/

via the provided middleware for inter-agent messaging. For the purposes of the SCE, a JADE application is created consisting of three primary agent types: Service Agents, Composition Initiator Agents, and Composition Terminator Agents. Other than these, JADE's default Directory Facilitator agent is used for the discovery of agents and the services they provide.

The *Service Agents* (SA) contain the primary mechanics of the SCE. In the current system each SA serves as a proxy for a single Web service. It is a simple matter to extend this model to a one to many relationship between an agent and several Web services, but the current system scales adequately and is better suited to demonstration and debugging. At startup each SA receives information describing the Web service it is to represent (from the Topic Map System), it then registers itself with the JADE Agent Platform's Directory Facilitator (DF). During this registration process, the SA uses the name of the Web service *operation* it represents as its service name. As the composition process executes, the DF is used by the SCE agents to perform service lookup based on registered service names. In this manner the DF is able to provide a mapping between a Web service operation name and the address of the agent serving as a proxy for the named service.

The *Composition Initiator Agents* (CIA) trigger searches for new compositions. When a composition request is sent to the engine, a unique CIA is created. Each CIA represents a virtual Web service that has no inputs, but whose outputs are the composition inputs specified in the composition request. The CIA starts the composition process by sending a composition request message to each Service Agent representing a Web service which consumes at least one of the CIA's outputs. Each composition computed by the engine will be rooted by a separate CIA agent, which can of course operate concurrently.

Composition Terminator Agents (CTA) are created to serve as an endpoint for successful compositions. Each CTA is matched with a unique CIA and as with the CIA, each CTA represents a virtual service. Its virtual service interface has inputs that match the desired outputs found in the composition request; CTAs produce no outputs. During SCE operation CTAs will wait to receive completed composition paths and publish them to an appropriate channel.

2.2 Topic Map System

In order to compose a collection of Web services, it is necessary that the interface to each Web service operation be available to the SCE. These interface descriptions are readily available in each service's WSDL file. In the WSDL file, one finds the Web service's operation name(s), the operation's associated input and/or output message names, and the constituent parts of each message. In order to compute a composition, this interface information is extracted from the WSDL files and made accessable to the SCE with a network addressable repository.

Rather than replicating the information from the WSDL files into a relational database, a Topic Map [4] representation was used. A Topic Map represents the definition of topics, the associations between topics, and links to occurrences of

the topic. Notably, the reason for the use of Topic Maps is that the composition engine is being enhanced to provide for semantically-driven service composition and referent metadata technologies, like Topic Maps, will be necessary to bridge between the various ontologies used for semantically describing Web services.

When discovering compositions, CIA and SA use a heuristic to prune the search space. The heuristic is based upon the definition of a *potential successor service*. A potential successor is a service that consumes one or more outputs of the current service. The agents acquire this information from the Topic Map System. Following is a discussion of the matching algorithm used by the SCE.

2.3 Matching Algorithm

The goal for determining compositions is simple: detect all possible invocation paths through a network of services connected by the input and output parts within their operation definitions. The foundation of our strategy consists of the architectural configuration previously described, where a single CIA-CTA pair is created per composition request and each WSDL description in the Topic Map is represented by an SA in the SCE. The SCE can essentially be modeled as a classical production system [5] which provides a computation model for solving a problem involving the notion of discovery. A generalized production system consists of three major components:

- One or more data repositories that hold information related to a given problem. Repositories may be static or dynamic.
- A set of production rules, generally in the form of (condition, action) pairs. When the condition is true, the rule is said to be enabled and the action thereby eligible to be performed. When a rule fires, its action is performed and the state of the repository is changed.
- A control strategy that specifies the order in which rules are evaluated and a conflict resolution mechanism which determines which rules to fire when several are enabled.

In the SCE, the repository of the production system is distributed across the Topic Map System, the platform DF, and the CIA, SAs and CTA. In general, for a given composition problem, the data found in the topic map and platform DF can be considered static. The SAs maintain both static and dynamic data in their local state. The static members of the SA's state consists of a production rule that *condition* specifies the collection of message parts that enable the underlying service, and the *action* contains the set of message parts output by the service. The dynamic state maintained by the agents of the SCE is always related to specific composition requests.

The SCE explores the search space of potentially connected services via the iterative building of a connectivity graph. Unlike many composition planning systems, knowledge of the potential connections between Web services is distributed across the local state retained by each agent. With this approach the problem is reduced from a question of composability, to one of reachability, i.e.,

is the CTA reachable by a path that extends from the CIA through a collection of SAs.

The dynamic information about potential invocation paths is shared amongst SAs via asynchronous message exchange. The SAs exchange composition requests messages, which alert an SA that they might be able to contribute to a composition. A composition request message, contains a construct known as an invocation path. The invocation path is always rooted at a CIA. The invocation path represents a valid composition chain from the CIA to the current service. It is the role of the SA to evaluate the invocation path, via its production rules in order to determine if it can contribute itself and its outputs to the building of a potential composition path.

The production rules and control strategy in the SCE define whether or not an SA can contribute toward an ongoing composition. A SA is responsible for evaluating its production rules against the contents of the invocation path found in a composition request message received from another agent (either CIA or other SA). The order of rule evaluation is as follows:

First, the SA checks to see if a received invocation path contains all input parts required to invoke the service. If so, then the service is considered *directly invocable*. This implies that all input parameters required by the service are available from predecessor services found in the invocation path.

Next, if the service is not directly invocable, a partial matching search is made of previously received invocation paths to determine whether a path conjunction can be formed. If this is the case the service is considered *conjunctively invocable*, as it is at the join of two or more split paths.

If the service is determined to be invocable, then the service is considered to be a candidate contributor to the current invocation path. As a candidate contributor, the SA must consider if it provides at least one output parameter that is not already available in the existing invocation path. If the *action* component of the enabled production rule contains a novel contribution to the invocation path, then the SA adds itself and its outputs to the path and sends composition request messages to its potential successors and the CTA. Recall that a potential successor is a service that consumes at least one output from the current service.

Finally, if it is determined that a service is not invocable, then the invocation path from the composition request message is retained within the local state of the SA. These non-invocable paths will be used by the SA at a later time when testing for conjunctively invocable paths.

Each SA performs this set of rules on each invocation path received. The CTA evaluates each composition request message it receives with the same algorithm used by the SAs, with the notable exception that if the CTA is invocable, it sends a notification that a valid composition has been discovered.

2.4 Concurrent Search

One major advantage of the distributed, multi-threaded approach taken is that multiple compositions can be computed in parallel. For each new composition request a CIA-CTA pair is created, with CIA loaded with the set of supplied

input parts and the CTA with the set of required output parts. To discriminate between the different compositions, we employ a reserved field in the ACL[6] messages exchanged between agents. In ACL parlance, the field is called the *conversation ID* and is intended for tagging messages belonging to the context of particular conversations (multiple message exchanges). We overload this usage slightly by using the field to identify invocation paths belonging to different composition requests concurrently working their way through the system. Each composition request is assigned a different conversation ID.

Notably, the distribution of a traditionally centralized algorithm requires thoughtful handling of search termination. The SCE handles termination by having the CTA timeout if an enabling composition path has not reached it within a configurable period of time. Each time a valid composition is detected by the CTA, the timer is reset. Upon timeout, the CTA notifies the SAs to stop searching for a connection path between the CIA and CTA for a particular conversation ID. Once an SA receives such a notification, it purges its local state of all invocation paths stored against the given conversation ID. Due to the distributed nature of the algorithm, notification of the CTA's timeout takes time to propagate across the population of SAs; thus the SA will discard all future messages it may receive for the given conversation ID.

3 Example of Operation

In order to demonstrate the operation of the service composition engine we will employ an artificial use case consisting of several services with a fixed interconnection pattern. The simple scenario consists of a set of WSDL defined services which define a set of operations existing within the context of a virtual manufacturing enterprise. Each service has a potential part to play in transitioning from the reception of a product order, through its manufacture to eventual delivery to a requesting customer. For the sake of simplicity, each service consists of a single operation requiring a fixed set of input parameters and producing a fixed set of output parameters. Although by no means intended to be a comprehensive model of a real system, this example serves as a suitable basis for providing an example within the space limitations of this paper.

A visualization of the connections between the services is depicted in Figure 2.

The composition request provides *order request* as input to the composition and requires *order delivery* as its output. An examination of the connectivity between the sample services shows that the composition engine should identify two composition pathways. One should indicate a path via Manufacturing Process A and another through Manufacturing Process B. The composition engine reports the two valid compositions; the output for the first composition is shown in Figure 3. The output provides the name of the CIA that initiated the composition, in this case CompositionInitiator-1. On subsequent lines, information about the

[6] ACL is the Agent Communication Language defined by the FIPA standards to which JADE is compliant

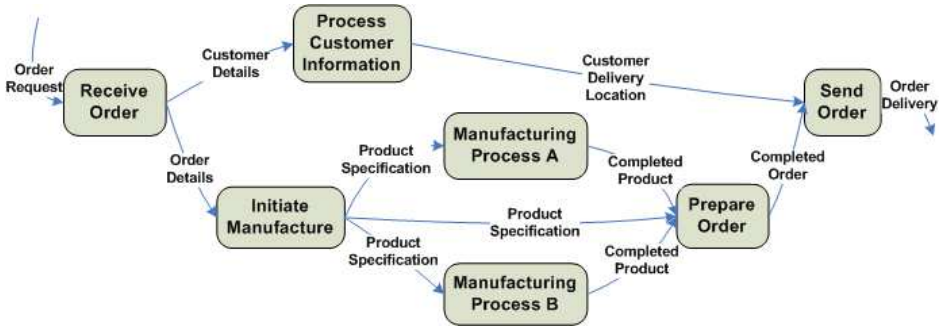

Fig. 2. A simple example of a set of services connections

services used in the composition is found, along with provenance information for each of the enabling inputs.

```
=> CompositionInitiator-1
     ---> inputs:
     ---> outputs:[order request]
=> ReceiveOrder
     ---> inputs:[order request] provided by [CompositionInitiator-1]
     ---> outputs:[order details], [customer details]
=> InitiateManufacture
     ---> inputs:[order details] provided by [ReceiveOrder]
     ---> outputs:[product specification]
=> ManufacturingProcessA
     ---> inputs:[product specification] provided by [InitiateManufacture]
     ---> outputs:[completed product]
=> PrepareOrder
     ---> inputs:[completed product] provided by [ManufacturingProcessA],
                 [product specification] provided by [InitiateManufacture],
     ---> outputs:[completed order]
=> ProcessCustomerInformation
     ---> inputs:[customer details] provided by [ReceiveOrder],
     ---> outputs:[customer delivery location]
=> SendOrder
     ---> inputs:[completed order] provided by [PrepareOrder],
                 [customer delivery location] provided by
                     [ProcessCustomerInformation],
     ---> outputs:[order delivery]
```

Fig. 3. First composition result from the initial request

4 Concluding Remarks

The coordination of Web services as a decentralized problem is a natural fit to real world situations where services are typically distributed, potentially across different semantic domains. As a result, we have selected a multiagent system approach to solving the composition problem due to its inherently distributed nature with strong coordination capabilities. This is of particular relevance when

service populations are dynamic and transient as is often the case with applications that span multiple domains and/or enterprise boundaries.

Our specific approach shows that a strategy consisting of local coordination guided by a globally applied configuration and matching algorithm is an effective means to solve the composition problem. Coordination between agents at the local level ensures the solution is easy to adapt and extend, while the global composition strategy adopted has been found to produce consistently accurate results. Detailed scalability studies need to be performed; however, by allowing SAs to represent multiple services and splitting the SA population across containers on separate hosts, we don't perceive scalability of the architecture to be a significant issue.

The work described in this paper represents the first step in a phased research plan. Importantly, the SCE is currently being extended to account for semantic Web service descriptions. Additionally, work is underway to provide a post processing mechanism which can convert the output from the composition engine into other representations, such as a graphical view or a WS-BPEL compliant process description.

Acknowledgments

The authors would like express their thanks to their respective organizations for the investment required to pursue this work.

References

1. Paul Buhler and José M. Vidal. Towards Adaptive Workflow Enactment Using Multiagent Systems. *Information Technology and Management Journal*, volume 6:1, 61–87, 2005.
2. Dominic Greenwood and Monique Calisti. Engineering Web Service - Agent Integration. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, The Hague, The Netherlands*, 2004.
3. P. Buhler and J.N. Vidal. Enacting bpel4ws specified workflows with multiagent systems. In *Proc. of the 2nd International Workshop on Web Services and Agent Based Engineering*, New York, U.S.A., July 2004.
4. ISO/IEC. Topic maps iso/iec 13250. Technical report, ISO/IEC, USA, 1999.
5. George F Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 5th Edition*. Addison-Wesley, 2005.
6. Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *LNCS*, volume 3387, pages 43–54. Springer-Verlag, 2004.
7. Collected Works. *EXP: Special Issue on JADE*, volume 3:3. Telecom Italia Laboratories, September 2003.

A Genetic Programming Approach to Support the Design of Service Compositions

Lerina Aversano, Massimiliano Di Penta, and Kunal Taneja

RCOST - Research Centre on Software Technology
University of Sannio
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy
{aversano, dipenta, tankunal}@unisannio.it

Abstract. During the last few years, Web service composition has received much interest to support business–to–business or enterprise application integration. Several approaches have been developed and are going on rather independently from each other. Nevertheless, the automatic/semiautomatic composition of web services is still an open research issue. In this paper we propose a search–based approach to support the design of service compositions. The approach uses genetic programming to automatically generate workflows that realize a functional goal.

Keywords: service composition, genetic programming

1 Introduction

Web services are rapidly gaining presence and popularity in the development of distributed software systems. However, while many schemas and standardizations have been proposed for collaborating processes of web services (also referred as service compositions), several open issues remain unresolved with respect to achieving an optimal solution. Considering that there are many web services exhibiting similar (if not equivalent) behavior, it is particularly relevant to properly compose the best set of services in order to accomplish a functional requirement.

Although the problem of finding the best way to compose services is relevant, many recent approaches essentially focused on the problem of single service discovery [1]. This because users dealing with the development of service–oriented system are, very often, interested to find operations that can be composed each other in order to accomplish a functional requirement. Indeed, one of the key promises of building applications with web services is that one should be able to compose a set of given services to create ones that are specific to the application's needs.

A number of XML–based standards have been developed to formalize the specification of web services, their automatic/semiautomatic composition and execution. Moreover, some research proposals have been presented. The service composition problem has been viewed as a planning problem in which state descriptions are ambiguous and operator definitions are incomplete. This brings back the problem to interpreting documents (which describe the world state), and introduces the need for a semantic type

matching algorithm. The matching algorithm together with an interleaved search and execution algorithm allow for basic automated service composition [2]. An algorithm for similarity research that exploits clustering techniques has been proposed by Dong et al. [3]. Their tool supports similarity search for web services, such as finding similar web service operations and finding operations that compose with a given one.

The need to automatically compose web service operations to solve a given problem has also motivated the research of Peer [4]. The paper follows the idea of using Artificial Intelligence (AI) planning software, but argues that the diversity of the web service domains is best addressed by a flexible combination of complementary reasoning techniques and planning systems.

In this paper we propose a search–based approach to support the design of service compositions. The approach uses genetic programming to automatically generate workflows that realize a functional goal. By using a Genetic Algorithm (GA) to aid in service composition, we aim to look into a search space considerably large, thereby obtaining a solution that is near–optimal given the computational constraints of GAs.

The rest of the paper is organized as follows. Section 2 describes the genetic programming approach for service composition. Section 3 illustrates, with a small example, how the approach can be applied. Section 4 concludes and outlines the work–in–progress.

2 The Approach

The objective of this work is to determine service compositions that realize a given functional goal. For example, if we consider as application domains those defined in the "context" ontology [5] we could be interested to build a service that receives as input an *Agent* entity and returns its personal information as output entities *Application*, *Phone*, and *Company*. A single service returning this information does not exist, but an aggregation of services can meet the requirement. The resulting workflow is represented in Figure 1.

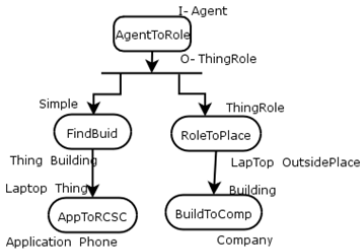

Fig. 1. Representation of Service Composition as a Tree Genome

Determining the best service compositions can be considered as an optimization problem: maximize a fitness function defined as the capability of the service composition of produce a given set of outputs once some inputs are available. To find a solution to this problem, different strategies can be adopted, for example AI planning algorithm, or Genetic Programming (GP). The next subsection provides a short primer on GAs, and GP in particular.

2.1 Genetic Algorithms and Genetic Programming

GAs originated with an idea, born over 30 years ago, of applying the biological principle of evolution in artificial systems. Roughly speaking, a GA is an iterative procedure that searches for the best solution of a given problem among a constant size population, represented by a finite string of symbols, named the genome. The search is made starting from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a fitness function and selected for reproduction using a selection operator. High–fitness individuals will have the highest probability to reproduce. The evolution (i.e., the generation of a new population) is made by means of two operators: the crossover operator and the mutation operator. The crossover operator takes two individuals (the parents) of the old generation and exchanges parts of their genomes, producing one or more new individuals (the offspring). The mutation operator prevents convergence to local optima, in that it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bit string). Further details on GAs can be found, for example, in the Goldberg's book [6].

In the case of GP, the genome represents a program (for example encoding the program's parse tree), while the crossover and mutation operators represent ways to recombine two programs in new one(s) or to create a variation of a given program. The fitness function measures how far the program encoded in the genome is from producing a given output. More details can be found in the Koza's book [7].

2.2 Encoding the Service Composition Problem with GP

To allow the GP algorithm search for a solution of our problem, we first need to encode the problem with a suitable genome. In our case, the genome is represented as a tree, where non–terminal nodes represent composite service workflow constructs, while terminal (leaf) nodes represent single services invoked inside the workflow. In other words, the genome may represent a simplified parse tree of a composite service, for example realized in BPEL4WS. Non–terminal nodes can be:

- *Sequence nodes*: where each node of the sequence produces outputs that are used by the subsequent nodes as inputs. For example, in Figure 2–a the service $S2$ uses the output $d2$ from $S1$, plus $d4$ (available from the root), to produce the output $d6$;
- *Switch nodes:* representing situations where the same output (goal) can be achieved in two different ways. For example, in Figure 2-b the output $d3$ can either be produced by $S1$ or by $S2$, that in addition also produces $d4$;
- *Flow nodes:* representing situations where more services can work in parallel to produce some outputs ($d3$, $d4$ in the example of Figure 2-c);

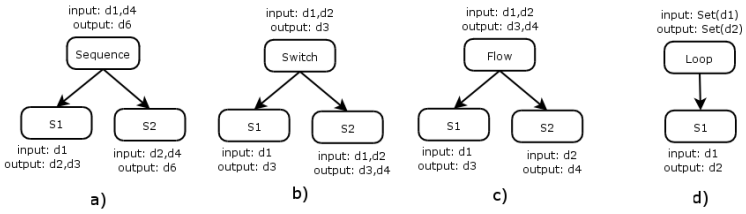

Fig. 2. Workflow constructs

– *Loop nodes:* where the input and/or the output are sets of data of a given type. In the example of Figure 2-d) iterations over the service $S1$ (that produces $d2$ from $d1$) give us a set of $d2$ starting from a set of $d1$.

Clearly, the above constructs only represent a rough model of the target workflow: loops needs to be completed specifying how composite data structure are handled and obtained, and how many times the loop has to be executed; much in the same way switch constructs need to specify the conditions. In our experience we believe that, to complete the production of the workflow there is still the need for human intervention, while the automatic, GP-based approach is able to select the services to be used and to give to the system integrator a rough idea of the composition itself.

Moreover, specific ontology concepts can refer to composite types and identify whole–part relationships. For example, a set of *books* can be referred as *library*. Thus, it can happen that our service requires a *library*, while a loop in the composition produces a set of *books*. In this case, reasoning over whole–part relationships is needed.

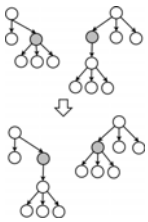

Fig. 3. One Point Tree Crossover

Let us now define the operators needed to let the GP operate. The crossover operator randomly selects two nodes (i.e., subtrees) from two individuals and swaps them, producing two new trees (individuals). An example is shown in Figure 3.

The mutation operator randomly picks a node and changes it in:

1. a non–terminal node, randomly selecting its type (sequence, switch, flow, loop) and adding to it some randomly selected (children), terminal nodes (i.e., single service

invocations). For sequence, switch and flow, the number of children is randomly chosen, while for the loop is always one; or
2. it changes the node in a terminal node, randomly selecting the invoked service.

Clearly, repeated applications of crossover and mutation operators ensure to produce trees of different width and depth.

Crossover and mutation are applied with probability 0.7 and 0.01 respectively. The GA used is the simple GA with non–overlapping population and with elitism of one (best) individual across subsequent generations (it means that the best individual is kept alive across subsequent generations). The selection operator is the roulette–wheel selection [6]: individuals with the highest fitness function have the highest probability to reproduce. The initial population consists of randomly generated trees of depth 2 with a max of 5 nodes as leaves.

Last, but not least, we need to define the fitness function for our problem. As we said, we need to generate a workflow that produces a set of outputs (goals) from a set of inputs. The far a solution is to achieving this goal, the worse should be the fitness function. We adopted the following criteria:

1. a correct solution, i.e., a workflow that produces the desired output from the available inputs, scores 1;
2. Partial solutions are evaluated as follows. The fitness of an individual is calculated starting from the leaves. If a particular leaf produces outputs desired by the parent, the score is incremented by 0.01.

The resulting formula is a function defined as (let us call $F_{goal}(g)$, where g is an individual's genome) :

$$F_{goal}(g) = w_1 * nums_c(g) + w_2 * nums_{nc}(g) + w_3 * \sum_{i=1}^{n} (num_{nodes}(g) + num_{loops}(g)) \tag{1}$$

where $nums_c$ is the number of solutions with correct arrangement of loops; $nums_{nc}$ is the number of solutions with incorrect arrangement of loops; n is the number of partial solutions (subset of a partial or a complete solution already in the individual is not counted); num_{nodes} is the number of nodes in a particular solution; num_{loops} is the number of correct loop in a particular partial solution. Finally, w_1, w_2 and w_3 are properly defined weights.

3 Illustrative Example

To show how the algorithm presented in this paper works we used a set of service interfaces defined by the authors, due to the lack of a significant number of service in public registries. The service interfaces specify the input/output parameters. Due to the limited space available, the list of services is not reported in this paper (the interested reader can access it from [8]). To make the examples more realistic, the service parameters are entities defined within ontologies. In particular, the examples that follow use the

entities defined in the "context" domain [5]. Let us now to consider the service requests reported in Table 1. Each request is specified by the input and the output of the service composition that an user aims to design.

	Inputs	Outputs
ServiceRequest1	Presentation, Person	Application, PDA
ServiceRequest2	Application	Person
ServiceRequest3	Simple, Presentation	Application, PDA, Phone

Table 1. Example of Service Requests.

The $ServiceRequest1$ specifies a request for compositions that receives as input the $Presentation$ and $Person$ entities, and returns its $Application$ and PDA. The second request is for compositions that returns $Person$ receiving $Application$ as input. The following Figures 4, and 5 show the service compositions found out by the algorithm considering the $ServiceRequest1$ and the $ServiceRequest2$ specified in Table 1.

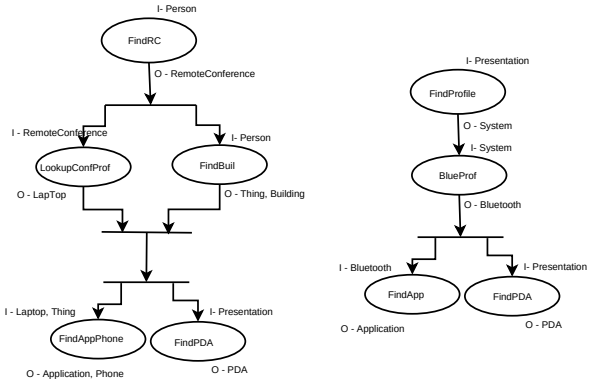

Fig. 4. Representation of Service Compositions found for ServiceRequest1

To evaluate the fitness function all the possible service compositions in the genome have been analyzed. In particular, as explained before, an individual scores 1.0 for every correct solution. Duplicate solutions, whenever present, do not score anything. The fitness of an individual is calculated starting from the leaves. If a particular leaf produces the desired outputs, the score is incremented by 0.01, and if the parent node produces the outputs required by this node another 0.01 is added to the score. Obviously, an individual may contain more solutions which satisfy the given input and outputs, and consequently the score of the individual increases with the number of solutions. For the

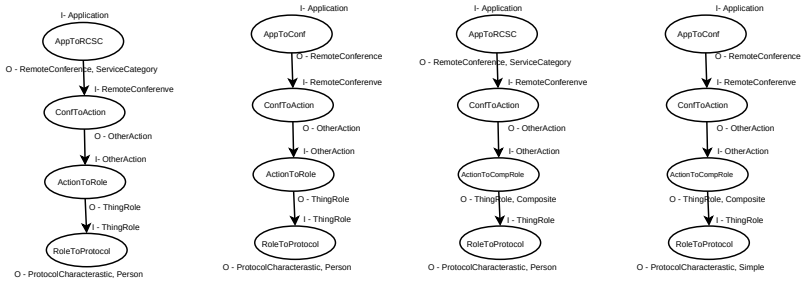

Fig. 5. Representation of Service Compositions found for ServiceRequest2

examples reported in this section, the defined weights are w_1= 1.0, w_2= 0.75 and w_3= 0.01.

For example, the genome represented in Figure 6 is found out by the algorithm considering the ServiceRequest3. It contains 4 service compositions. The first one, composition 1 in Figure 6, scores 1.0 since it produces exactly the required outputs (Applications, PDA) from the given inputs. The second composition, composition 2 in Figure 6, is a subset of the third composition (composition 3 in Figure 6), therefore it does not score anything. While in the third composition the leaf node produces the outputs required and the parent of the leaf node produces the inputs required by the leaf but partially fails the input condition. As a consequence, it scores 0.02, since its a partial solution till 2 nodes. The fourth composition fails to produce the outputs so it does not score anything. Then the overall genome scores 1.02 (as sum of the single composition score). The results of the algorithm have been compared with the results of an exhaustive search approach. We found out that the GP was quite successful in finding a solution up to 5–6 levels of tree depth as compared to exhaustive search. Moreover, we experienced that it was not possible for exhaustive search to find a solution even in a large number of generations.

4 Conclusions and work–in–progress

This paper presented an approach based on Genetic Programming that creates service compositions starting from i) a set of inputs and desired goals, and ii) a set of available services. Genetic Programming resulted effective to explore a large search space and properly combine the most suited services.

The resulting workflow indicates the composition structure in terms of sequences, flows, switches, loops. Such a workflow is not directly executable: loop and switch controls expressions need to be manually specified, and also manual intervention is needed to handle recovery actions [9]. Nevertheless, the obtained workflow constitutes a potential guideline for the service integrator.

Work–in–progress is devoted to apply the approach to more complex case studies, involving sets of real services, and to assess its usefulness with controlled experiments.

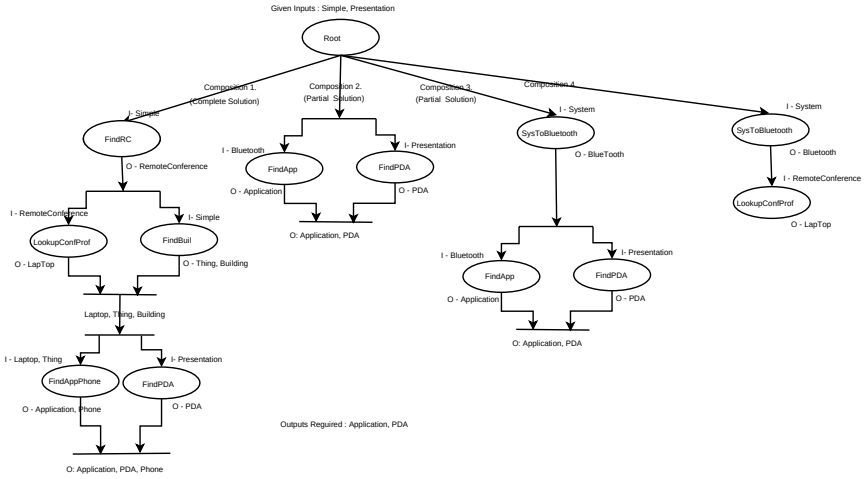

Fig. 6. Representation of Service Compositions found for ServiceRequest3

Last but not least, the approach needs to be further improved, for instance with the capability of dealing with service non–functional properties, and of handling constrained inputs and outputs.

References

1. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: First International Semantic Web Conference (ISWC 2002). Volume 2348., Springer-Verlag (2002) 333–347
2. Carman, M., Serafini, L., Traverso, P.: Web service composition as planning. In: Workshop on Planning for Web Services in conjunction with The 13th International Conference on Automated Planning Scheduling. (2003)
3. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: Proceedings of the 30th International Conference on Very Large Data Bases. (2004)
4. Peer, J.: A PDDL based tool for automatic web service composition. In: Proc. of the Second Workshop on Principles and Practice of Semantic Web Reasoning at The 20th International Conference on Logic Programming. (2004)
5. Khedr, M.: Context ontology (2005 (last accessed)) *http://www.site.uottawa.ca/ mkhedr/contexto.html*.
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Pub Co (1989)
7. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). The MIT Press (1992)
8. Aversano, L.: Service interfaces (2005 (last accessed)) *http://www.rcost.unisannio.it/aversano/servicelist.html*.
9. Hagen, C., Alonso, G.: Exception handling in workflow management system. IEEE Transactions on Software Engineering **10** (2000) 943–958

A Classification of Issues and Approaches in Automatic Service Composition

Ulrich Küster,[1] Mirco Stern,[2] Birgitta König-Ries[1]

[1] Institute of Computer Science, Friedrich-Schiller-Universität Jena, D-07743 Jena, Germany, ukuester|koenig@informatik.uni-jena.de
[2] Institute for Program Structures and Data Organization, Universität Karlsruhe, D-76128 Karlsruhe, Germany, mirco.stern@stud.uni-karlsruhe.de

Abstract. Resulting from the constantly increasing usage of service oriented computing, the problem of automatic composition of services is rapidly gaining attention. Many solutions have been proposed, yet too little attention has been paid to thoroughly analyzing the different needs that require the ability of (automatic) service composition. This paper therefore aims at identifying the different cases in which service composition is needed and at classifying existing approaches accordingly. Three distinguishable types of service composition applications are described along with their basic problems and various approaches addressing the identified classes are presented.

1 Introduction

Automatic service composition remains one of the key challenges of service oriented computing today. In general, "service composition" can be defined as creating a composite service, obtained by combining available component services. It is used in situations where a client request cannot be satisfied by any single available service, but by a combination thereof [1]. In terms of software engineering automatic service composition will significantly enhance the power of service oriented architectures (SOAs). SOAs combine available base services in order to build higher level services or distributed applications. Specifying manually which base services to use and how to combine them is a cumbersome task. Furthermore the resulting composite process is vulnerable to change as it must be corrected manually when base services become unavailable or new better services appear. Automatic service composition has the potential to dramatically change the way how SOAs are engineered: Given a rich set of base services and efficient, reliable automatic service composition methods the vision of programming as specifying *what* a program is supposed to do and not *how* it is supposed to do it can become reality. That way software engineers could create flexible programs, able to adopt to changes in the environment completely autonomously.

Service composition in general can be differentiated into *synthesis* and *orchestration*. *Synthesis* refers to *generating a plan* how to achieve a desired behavior by combining the abilities of multiple services. In contrast, *orchestration* refers to *coordinating the control and data flow among the various components* when

executing that plan. While our focus in this paper is on automatic synthesis, orchestration is an important problem that is complementary to synthesis. Examples of "service composition" approaches referring to orchestration include [2, 3]. Concerning synthesis the definition of service composition given above covers a number of different problems with distinct characteristics, each of which has been addressed by different approaches. This work's goal is to (a) identify the distinct classes of composition applications, (b) point out specific problems of each of those classes and (c) give an overview of existing approaches and their suitability for the individual classes.

A number of surveys on service compostion exist, however, none of these meet the goals listed above: Rao et al. [4] provide a quite comprehensive overview categorized according to the technique used, i.e. workflow techniques and various AI planning methods. Another survey solely centered around AI planning is [5]. In a more general overview Benatallah et al. [6] focus on workflow-based approaches for web service integration. Some of the considered approaches are abstracted in the form of software design patterns. Another survey on web service composition platforms is [7]. Milanovic et al. [8] provide an overview of different approaches for modelling composite services that are evaluated against a number of requirements to service composition modelling.

In contrast to all these surveys, this paper is structured around a classification of different *applications* of service composition and furthermore specifically focuses on *automatic* service composition. Given a service request that can't be met right away those existing automatic composition approaches are introduced that seem best suited to overcome the specific type of problem at hand. We believe that a categorization along this domain is the most helpful one from an engineering point of view. In detail we identify the following three classes of problems:

- **Fulfilling preconditions**: A service that can provide the desired effects exists, however, not all of this service's preconditions are met from the outset.
- **Generating multiple effects**: The request encodes multiple effects that are related, yet can be generated by different services.
- **Overcoming a lack of knowledge**: Additional knowledge gathering is necessary in order to correctly fulfill a request.

In the following Sections 2, 3 and 4 we elaborate on these different classes of service composition needs. In Section 5 we give a summary and conclude.

2 Fulfilling Preconditions

In an attempt to find an appropriate service providing a requested functionality it is a common situation to find a service that is able to generate the requested effects but some of its preconditions are not fulfilled. In these cases, "chaining" services can be used to overcome that problem. There might, e.g., be a request for a service that translates a given document from Chinese to German. If no such service exists a service that translates English documents to German can be used

in conjunction with another service that translates from Chinese to English. Such chaining can be understood as a way to compose services by recursively regarding the effects of one service as the preconditions of a following one until a desired effect is reached. The most common approaches to service chaining include graph search, forward chaining, backward chaining and estimated-regression planning. Each of them will be considered in turn.

Graph Search approaches rely on building a graph representation of all services available. In [9], Zhang et al. build a graph whose nodes represent the available services and whose edges encode whether one of a service's outputs may serve another service as one of its inputs. Edges are weighted according to the level of correspondence of the associated input and output. An adapted Bellman-Ford algorithm is used to find the shortest paths from the user's inputs to the expected outputs which represent the best available composition. The same idea has been used in an earlier work [10] with the restriction that services are only allowed to have a single input and a single output which simplifies the graph and the corresponding searching algorithm. A problem of this class of approaches is that they do not scale well with the number of offered services. For instance, the algorithm by Zhang et al. works in $O(n^3)$.

Forward Chaining is typically used by approaches rooted in logic-based planning systems. Starting with the available knowledge about the world and the preconditions and inputs from the service request the planning system uses applicable services, i.e. services whose preconditions can be met, in order to infer additional knowledge until the requested effects are fulfilled. An example of this approach is SWORD [11]. Another example is [12] which also exhibits the fundamental problem that forward chaining has: its undirected search. While the requested effects always have to be generated completely, in many cases it is not necessary to use all knowledge about the world in order to accomplish this. Thus, forward chaining tends to search in directions that are unnecessary for the requested effects[1].

Backward Chaining overcomes this problem. Backward chaining works similar to forward chaining with the difference that the composition starts with those services generating the requested effects instead of those whose preconditions are grantable. Any algorithm then recursively tries to create the necessary preconditions using other services until all preconditions are met. In [13], Sirin et al. use a semi-automatic approach to establish the composite service. Here, the user decides which service to add to the chain whenever more than one possibility exists. An example of a fully automatic algorithm using backward chaining is [14] where Sheshagiri uses heuristics[2] to pick the next service in the chain whenever there is a choice. While backward chaining generally implements a better directed search than forward chaining the problem space typically is still large.

[1] [12] tries to overcome this problem by including search control in the deduction rules of their planner.

[2] To give an example, one of the proposed heuristics is to use the service with the smallest number of preconditions.

Estimated-Regression Planning is an attempt to improve the performance of the search by implementing a forward chaining guided by a heuristic based on backward chaining. McDermott [15] applies this idea to the domain of web services. While he doesn't actually compose services but rather plans the interaction with a single web service (i.e., he plans an execution sequence of atomic interactions leading to the requested result) his results are applicable to the domain of service composition as well.

Concluding remarks. Generally, when implementing automated chaining of services, it should be noted that chaining may insert a degree of uncertainty regarding the semantic correctness of the generated composite service. In theory, a service that translates Chinese to German is equivalent to a composite service that translates Chinese to Japanese to English to German. In practice, each translation involves a loss of accuracy and, thus, the two services are far from equivalent. Although these conversion tasks are a natural application of chaining, to the best of our knowledge this problem hasn't been addressed so far.

Another problem of service chaining are cases where a user does not want a precondition of a service to be fulfilled due to side effects of that action. Take for instance a service that requires the user to have a valid credit card. Assuming one doesn't have a credit card a chaining algorithm might try to create a credit card contract in order to overcome this problem. Obviously that might be considered unacceptable. Consequently, a user's preferences need to be regarded when designing automated service chaining algorithms.

As a final remark it is noted that a large number of approaches to chaining is based on AI planning systems which generally leads to scalability problems in case of an increasing number of available services.

3 Generating Multiple Effects

Service requests like "I want to travel to Italy and need a hotel and a flight to a nearby airport" are characterized by the fact that multiple effects (flight and hotel booking) are requested. These effects are related (e.g. by the travel dates and by a shared location) but typically can be fulfilled by different services.

Chaining is used to provide a requested functionality that typically could be provided by a single service which simply is currently not available. In contrast, when dealing with multiple effects, service composition is somewhat more inherent to the request. Furthermore, when multiple effects are specified, the problem of composition synthesis is really a problem of decomposing the request into a collection of component requests each of which can be satisfied by a single service. Basically such service composition approaches can be divided into two categories. These categories differ in the granularity at which services are regarded. The first category, *behavior-based service composition*, relies on services being described by the their messages and the possible sequences thereof, i.e., the interaction with a service. In contrast, in the second category, *component-based service composition*, services are regarded as atomic, that is they can only be executed as a whole. Thus, in the first case, it is possible to combine parts of

services to new services, whereas in the second approach, only complete services can be combined to new ones. In addition to exploring approaches in these two categories, a closer look at at special subclass of requests with multiple effects, namely quantified requests, is worthwhile and will be provided later on in this section.

Behavior-Based Service Composition. Most approaches for modelling behavioral descriptions, especially those targeted towards composition, are based on finite state machines [16]. As an example, in [1], Berardi et al. describe all available component services as well as the request as finite state machines. Their goal is to find a composition of offered services that permits the interaction sequences given by the request. By using DPDL (Deterministic Propositional Dynamic Logic) the problem of service composition is reduced to the problem of satisfiability of a constructed formula. One of this approach's problems is that it doesn't scale well (needs $EXPTIME$). A related approach is described by Bultan et al. [17].

Component-Based Service Composition. The second category can be characterized by the use of decomposition frameworks that directly contain the component services to be used. A large body of research in this field uses some sort of rules to manage the decomposition of requests in a more or less automated fashion. As an example, Wu et al. [18] have implemented a hierarchical task network planner whose knowledge base contains methods representing the decomposition of higher level tasks. Similarly, McIlraith et al. [19] encode the decomposition using constructs of the logic programming language Golog, which is used for creating a sequence of services providing the requested functionality[3].

An approach based on data integration techniques is proposed by Thakkar et al. [20]. Here, the decomposition is implied by regarding the available services as views over a global schema. As a consequence, only information retrieval requests can be supported.

The work by Medjahed et al. [21] is an example of an approach in which the composition is contained in the client's request, given in a proprietary specification language CSSL (Composite Service Specification Language). Here, the exact sequence of desired operations (or to be more precise, of *descriptions* of the desired components) is specified in the request. The main contribution of this work is a composability model which captures information about whether available services that match the components' descriptions can be combined as specified in the request.

Quantified Requests. A special case of requests asking for multiple effects are quantified requests like "I want to download *all* publications of a certain author" where the same effect (download a publication) is requested multiple times. Such a request might be answered by a single service if that source has all publications available (e.g. a publication database) but often it will be neces-

[3] An interesting point about this work is that additionally chaining can sometimes be used to fulfill the preconditions of a component service.

sary to query multiple services (e.g. all common publishers) and to combine the results.

This kind of requests should not be confused with a request for "all services" providing a certain effect, like all services offering publications by the author. From a procedural point of view the difference is that in the case of a request for all services, the problem is in locating services, whereas in our case the problem is in composing services. From a semantics point of view, the result of the first request could contain duplicates of the same publication (provided by different service providers), which is not possible in the second case.

Here, a request's effect (or an effect's attribute) is quantified and the problem is to combine various services in order to cover the requested publications. The challenge is first to get to know which publications to look for and second to combine the available sources in an optimized way. To the best of our knowledge, this problem hasn't been addresses so far regarding service composition, even though it contains a large number of quite natural requests. There are approaches available that probably could be extended to cover this case, e.g. those rooting in first order logic planning systems like [12] but this remains as future work.

Concluding Remarks. One problem that needs to be dealt with when using composition in order to generate multiple effects is how to enforce transactional properties when invoking multiple services. In the context of the travel example one probably doesn't want to book a hotel anymore, if the booking of the flight fails. This problem has been addressed, e.g., by Casati et al. [22] or by the Web Service Transaction Specification [23]. It should also be noted that especially some of the earlier approaches listed above are based on semantically weak service descriptions (the same is true for approaches addressing service chaining). More recent approaches have broadened their focus to address this problem.

4 Dealing with Missing Knowledge

When searching for an appropriate service it may happen that knowledge necessary to choose the correct service is missing. Taking up the traveling example once more, assume the requested service is supposed to book a flight given a maximum price. Assume further that there is a service available booking flights with a certain airline by their flight number. This offer could be used – provided one knows how much each flight costs. Moreover, even if one has this knowledge, one still needs to know which flights are available.

But since assuming complete knowledge is obviously unsuitable for the Semantic Web [24], it is a basic requirement of any approach that aims at finding a (maybe composed) service matching a certain request that this approach is able to deal with incomplete knowledge. That is, the algorithm must be able to detect such a situation and implement procedures to acquire the missing knowledge.

One can distinguish two approaches addressing this problem: (a) one can make use of knowledge services **while searching** for a suitable service or (b) this search can be deferred until the time of execution. In this case a **conditional plan** has to be established. Note that in the first case, even if the planner returns

a single service sufficient for providing the requested effects, we consider this to be a form of service composition as the cooperation of multiple services is necessary and is only brought forward to planning time.

Creating Conditional Plans. Here, the knowledge gathering services are executed in conjunction with the remaining services and are used to navigate through the plan. In [15] McDermott proposes an extension of DPDL to model gaining information (and thereby to overcome the restriction of relying on perfect knowledge). Whenever the planner encounters a situation in which it lacks knowledge that can be learned it inserts an information gathering action into the plan. Branches are created for different outcomes of the information retrieval. The approach for building conditional plans taken by Martinez et al. [12] is based on a similar idea. One advantage of using conditional plans is that the time consumption at planning time of the alternative approach is dominated by the time spent waiting for knowledge gathering services to complete their execution which can be arbitrarily long. A further advantage is that knowledge services that have world altering effects like causing costs might not be tolerated at planning time. Finally, if one gathers information at planning time, this information may have become outdated at execution time [19][4]. The disadvantages of using conditional plans include a sometimes unrealistically high number of alternative paths. In the flight booking example for instance, one would have to create a path for each available service offering flights to the target location in case that all other services are booked up. In this case the number of paths is determined by the number of available services. This doesn't scale well. Additionally, having to consider a large number of alternative paths leads to an increased time consumption when creating the plan. These disadvantages lead to approaches that retrieve information at planning time.

Searching While Planning. SHOP2 [18] is an example of this approach. Wu et al. designed their planner's decomposition rules such that the planner is able to distinguish between component services gathering information and others, so that the former ones can be executed at planning time. While in this work the mechanism to determine *when* to ask for *what* information can be regarded as "hard wired" in the rules, in a follow up work the authors remedied this leading to automatic recognition that information is missing [25]. An approach to gathering information similar to that of SHOP2 is taken by McIlraith et al. [19].

Concluding Remarks. A fundamental problem of approaches aiming at *automatically* recognizing that information is missing: there needs to be a mechanism to determine whether the available knowledge is complete or not. In the flight booking example it would be necessary to know when the found set of flights is complete to be able to pick the cheapest flight with certainty. The problem here is to know when to stop searching for more information. This problem has been, for instance, addressed by [24].

[4] McIlraith et al. address this problem using a monitor that repeats the knowledge gathering at execution time.

5 Summary and Conclusion

In this paper we have analyzed the different needs that require the ability to (automatically) compose services and have identified three distinct classes:

First of all, service chaining addresses the problem of *fulfilling a service's preconditions* by using another service. Solutions to this problem have to take into consideration that the chaining of services may insert a degree of uncertainty and that automated chaining is sometimes unacceptable for the user.

A second class of service composition approaches aims at *generating multiple effects* that are somehow related. As a special case we identified requests containing a quantification of some attributes. Problems that have to be addressed by approaches of this class as well as by approaches of the former one include ensuring scalability in the number of offered services and establishing transactional properties of the service composition.

Finally, the problem of *incomplete knowledge* can be solved using service composition by adding knowledge gathering services to acquire it. The main difficulty here is how to decide when to stop searching for additional knowledge.

Our overview has shown that while quite a number of approaches to service composition exist, it is necessary to take a close look at the requirements of ones particular situation to find a suitable approach. Also, while many problems concerning service composition have been solved, there remain important open issues that need to be addressed in order for service composition to become a feasible tool. Among the most important of these issues are the degrading quality in long service chains, the lack of transactional properties of composed services, scalability with an increasing number of offered services, the foundation of most approaches on semantically weak service descriptions, the lack of approaches to allow for quantified requests and an appropriate treatment of missing knowledge.

References

1. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: Proc. of 1st Int. Conf. on Service Oriented Computing (ICSOC-03), Trento, Italy. (2003)
2. Benatallah, B., Sheng, Q.Z., Ngu, A.H.H., Dumas, M.: Declarative composition and peer-to-peer provisioning of dynamic web services. In: Proc. of the 18th Int. Conf. on Data Engineering (ICDE'02), San Jose, CA, USA. (2002)
3. Pistore, M., Bertoli, P., Barbon, F., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Proc. of the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS 2004), Whistler, BC, Canada. (2004)
4. Rao, J., Su, X.: A survey of automated web service composition methods. In: Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition, SWSWPC2004, LNCS, San Diego, USA. (2004)
5. Peer, J.: Web service composition as AI planning - a survey. Technical report, Univ. of St. Gallen, Switzerland (2005)
6. Benatallah, B., Dumas, M., Fauvet, M.C., Rabhi, F.A.: Towards patterns of web services composition. In: Patterns and skeletons for parallel and distributed computing, London, UK, Springer-Verlag (2003) 265–296

7. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web and Grid Services **1** (2005) 1–30
8. Milanovic, N., Malek, M.: Current solutions for web service composition. Internet Computing, IEEE **8** (2004)
9. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: Proc. of the 2003 Int. Conf. on Web Services (ICWS'03), Las Vegas, NV, USA. (2003)
10. Mao, Z.M., Brewer, E.A., Katz, R.H.: Fault-tolerant, scalable, wide-area internet service composition. Technical Report UCB//CSD-01-1129, University of California, Berkeley, USA (2001)
11. Ponnekanti, S.R., Fox, A.: SWORD: A developer toolkit for web service composition. In: Proc. of the 11th Int. WWW Conf. (WWW2002), Honolulu, HI, USA. (2002)
12. Martínez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: Proc. of the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS 2004), Whistler, BC, Canada. (2004)
13. Sirin, E., Hendler, J.A., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Proc. of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI'03), Angers, France. (2003)
14. Sheshagiri, M.: Automatic composition and invocation of semantic web services. Master's thesis, University of Maryland, Baltimore County, USA (2004)
15. McDermott, D.V.: Estimated-regression planning for interactions with web services. In: Proc. of the 6th Int. Conf. on Artificial Intelligence Planning Systems (AIPS'02), Toulouse, France. (2002)
16. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: a look behind the curtain. In: Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'03), San Diego, CA, USA. (2003)
17. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: Proc. of the 12th Int. Conf. on World Wide Web (WWW'03), Budapest, Hungary. (2003)
18. Wu, D., Parsia, B., Sirin, E., Hendler, J.A., Nau, D.S.: Automating DAML-S web services composition using SHOP2. In: Proc.of the 2nd Int. Semantic Web Conf.(ISWC2003), Sanibel Island, FL, USA. (2003)
19. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France. (2002)
20. Thakkar, S., Knoblock, C.A., Ambite, J.L.: A view integration approach to dynamic composition of web services. In: Proc. of the 13th Int. Conf. on Automated Planning and Scheduling (ICAPS'03), Trento, Italy. (2003)
21. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. The VLDB Journal **12** (2003) 333–351
22. Casati, F., Ilnicki, S., jie Jin, L., Krishnamoorthy, V., Shan, M.C.: Adaptive and dynamic service composition in *eflow*. In: Proc. of the 12th Conf. on Advanced Information Systems Engineering (CAiSE*00), Stockholm, Sweden. (2000)
23. IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA: Web service transaction specification (2005)
24. Heflin, J., Munoz-Avila, H.: Lcw-based agent planning for the semantic web. In: Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI2002), Edmonton, AB, Canada. (2002)
25. Kuter, U., Sirin, E., Nau, D.S., Parsia, B., Hendler, J.A.: Information gathering during planning for web service composition. In: Proc. of the 3rd Int. Semantic Web Conf. (ISWC2004), Hiroshima, Japan. (2004)

An Ontology for Quality-Aware Service Discovery

Steffen Bleul and Thomas Weise

Kassel University, Distributed Systems
{*bleul, weise*}*@vs.uni-kassel.de*

Abstract. The fast emergence and acceptance of service oriented architectures leads to fast development of extensional technologies like service delivery, discovery and composition. As main effort is being spent on automatic discovery and composition, current solutions do not reflect real world scenarios sufficiently. Services are offered by different vendors with different quality levels and prices. Large service oriented architectures with dynamic service compositions are not able to adapt without manual inspection of service quality and negotiation of service contracts. We propose an ontology for modelling Quality of Services (QoS) and Service-Level-Agreements (SLA). A semantic approach should bridge the gap of different terminology, languages and metrics making Service-Level offers and requests agent understandable and automatic quality-aware discovery possible.

1 Introduction

Service oriented architectures (SOA) have become important means for implementing enterprise processes in the industry and on the web. Service oriented architectures are potentially easy to maintain, to extend and to improve. Services are building blocks in implementing business processes in companies and to integrate heterogeneous resources and external systems [BHMS05]. Therefore a SOA is a powerful software design paradigm.
Multiple services are necessary to realize a process which is usually represented by a workflow [RS04]. Companies implement processes like ordering, billing and accounting by composing several services where each service realizes a process task and services communicate with messages. Composition of services is supported by a process execution language like BPEL4WS [IBM03], and a message choreography language like WS-CDL [Wor04]. These languages allow automatic execution of service compositions because they allow the specification of workflows of services and message choreography between services.
Service oriented architectures are easy to evolve by replacing each service with a better substitute. Substitutes may offer less response time or traffic. Especially outsourced services offer flexibility in choosing software vendors or providers of specialised services, e.g. providing an improved search function for the internal infrastructure. Therefore it is beneficial to use service discovery mechanisms to adapt service compositions to changing quality and pricing.

Although automatic service discovery is already possible, e.g. with IOPE matching algorithms, like the one presented in [JT04]. However it currently does not take into account quality and pricing aspects of services. Using outsourced services also demands special actions like dealing out service level obligations, contract signing, accounting and quality monitoring. Based on current SLA (Service Level Agreement) description languages like WSLA [ea02] and WSML [SDM02] we believe a semantic approach can automate quality aware discovery of services. We propose an ontology for machine understandable service level requests and offers for quality-aware service discovery. As a result it enables dynamic binding of external services in an internal workflow under quality and pricing aspects. Therefore the ontology contributes to service brokering and dynamic service bindings in large-scale service oriented architectures and compositions.
The remainder of this paper is structured as follows. The next section is an introduction to quality-aware service discovery and presents a use case. Section 3 is an overview of the proposed ontology and section 4 gives an example on matchmaking in quality-aware service discovery. Section 5 is about related work before the paper closes with a conclusion.

2 Quality-Aware Service Discovery

Quality-aware discovery of services comprises finding a service with the requested quality dimensions and checking if the service can offer the requested guarantees. Even if service descriptions offer specifications of service dimensions and guarantees, quality-aware service discovery remains cumbersome. The task is even harder if different parties use different terms and metrics for the same quality dimension. Furthermore, providers can offer different service levels at different prices and enhance certain quality levels by offering supplementary service packages. A service package is a union of obligations and its price. The user can combine service packages and supplementary packages to accomplish his requested service level.
A quality level consists of several obligations. Each obligation offers a guarantee on a quality dimension, e.g. a guarantee on response time with at most 200 milliseconds. Service provider and service user both negotiate a certain quality level by agreeing on obligations on quality dimensions. The outcome is a SLA with several obligated SLO (Service Level Objectives). Service Level Agreements are necessary for continuous service delivery and quality. The providing party must enforce these objectives or face negotiated violation penalties like contract cancellation or paying violation fees.
In Figure 1 we present an example for a service level offer on the left and service level request on the right side. The service level request consists of three guarantees on the quality dimensions response time, transactions and contract time. A service provider offers the necessary service for travel booking with guarantees on the requested dimensions but two kinds of service levels. The Gold-Guarantees are better than the Silver-Guarantees but also at a higher price. A supplementary package offers enlarging the contract time for one month at a price of five

Service Offer Travel Booking Service				Service Request Travel Booking Service	
Quality-Dimensions	Gold-Guarantees	Silver-Guarantees	Supplementary-Package	Quality-Dimensions	Requested-Guarantees
Responsetime	< 200ms	< 400ms		Responsetime	< 0,4 s
Transactions	10000	5000		Transactions	> 4000
Contract time	12 month	12 month	+1 month	Contract time	1 year
Price	50 €	20 €	5 €		

Fig. 1. Example scenario

Euros. As you can see, the offered guarantees on response time are measured in milliseconds but the requested is measured in seconds. So they have semantically the same dimensions but different metrics.

Existing service descriptions allow searching for syntactically matching services. Discovery of services is done manually by browsing search results for semantically matching services which is cumbersome in large registries. There is not only the need for syntactic description languages but also for semantic interface descriptions. Currently this is realized mostly by the WSML [SDM02] and OWL-S [ea04] ontologies. OWL-S is based on OWL (Ontology Web Language) the W3C proprietary language for the Semantic Web [MH04]. Both languages allow the semantic description of Web Service interfaces.

3 Service Level Ontology

As already mentioned, current service brokering approaches lack the ability to model service level packages and pricing. In the real world service brokering is much more flexible. Service providers offer service level packages, e.g. gold, silver or copper packages, with different pricings and service level objectives. A service level package can be optimized by buying supplementary packages, e.g. buying additional transactions or a better guarantee on response time.

Otherwise quality-aware service discovery is limited to matching obligations on quality dimensions against each other. This decreases the probability of successful service discovery. Our approach proposes a SL-Ontology (Service Level) for modelling service level packages, supplementary packages and pricing for service offers. It includes modelling of QoS, metrics and units. The following paragraphs give an overview of our SL-Ontology followed by an example on matchmaking in the next section.

The structure of our SL-Ontology can be illustrated as an ontology pyramid, which is presented in Figure 2. On top we distinguish between three individuals of the SL-Ontology. The *Service-Level-Offer* concept represents the service level provider and his offered service level packages. The *Service-Level-Request* concept represents a requested obligations for the wanted service level and finally

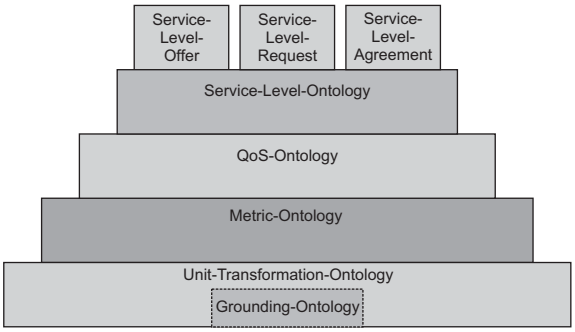

Fig. 2. Service-Level-Ontology Pyramid

the *Service-Level-Agreement* concept specifies the service level both parties agree on. While current service discovery does not distinguish between offers and requests, we believe that differentiation between them is necessary when handling multiple requests and offers in one knowledge base.

The *SL-Ontology* is build on top of a *QoS-Ontology*. The *QoS-Ontology* is used for modelling service quality dimensions. Since different providers can use different identifiers for the same quality dimension, the ontology allows binding of individuals to external taxonomies. Therefore we can distinguish between dimensions, which relate to semantically the same concept. Each quality dimension value is defined by a metric. The *Metric-Ontology* specifies the unit and data type the provider uses to measure his quality dimension. We have to know the data-type of the quality dimensions. Otherwise if we use a data type that is too limited, we will have inaccurateness by interpreting the quality dimension value. At the bottom of the ontology pyramid the *Unit-Transformation-Ontology* is the base ontology for semantic specification of service levels. Each metric has its unit. The most obvious problem in matching service dimensions is the use of different units for the same metric. Response time for example can be measured in milliseconds or seconds. Therefore the *Unit-Transformation-Ontology* allows specifying the functional relation between different units. The *Grounding-Ontology* is used for grounding semantic concepts on logical entities like rule descriptions for guarantees or service execution for unit transformation.

4 Quality-Aware Matchmaking

Let us look at matchmaking between a Service-Level-Offer and a Service-Level-Request. Matchmaking as part of a discovery process will show how our ontology enhances quality-aware service discovery. We start to illustrate how to specify individuals with our SL-Ontology and realize matchmaking between them. We use the example of section 2, Figure 1.

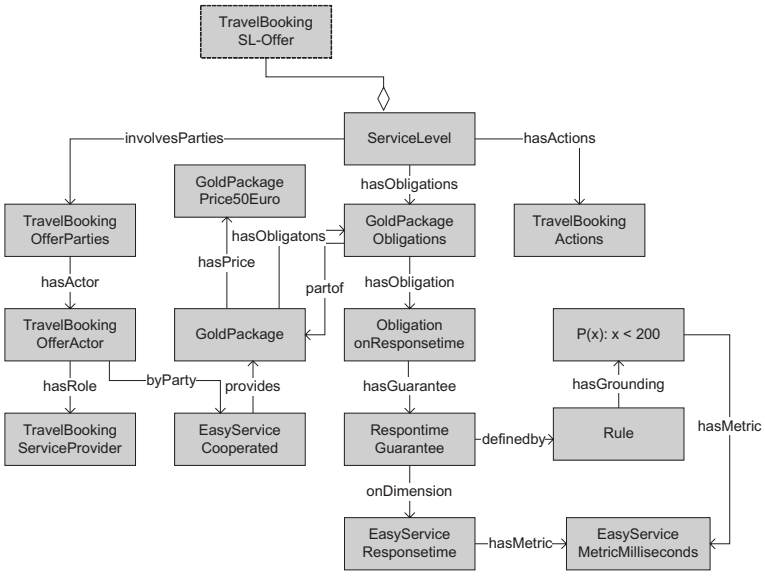

Fig. 3. Service-Level-Ontology Example

In Figure 3 we illustrate an individual of the SL-Ontology. The individual is a SL-Offer of the Travel Booking service. The service provider is the Easy Service Company which is specified by the *EasyServiceCooperated*-Individual. The service level obligations are part of the gold package. This is specified by the *partof*-Property between the *GoldPackage*-Individual and the *GoldPackageObligations*-Individual. The other packages are modelled analogously to this example. Our semantic matchmaking example of quality dimensions consists of three parts:

1. We have to decide if any offered package deals with guarantees on the same quality dimensions as the requested quality dimensions. Otherwise the package can not achieve at least one requested quality guarantee.
2. Check if two semantic equivalent quality dimensions use different metrics.
3. Check if the offered guarantee satisfies the requested guarantee.

As a matter of simplicity we limit the example of matching guarantees on response time and leave out the effects of supplementary packages. Actions for contract signing and contract cancellation are matters of service choreography or orchestration and therefore out of the scope of this paper. Obligations consist of a guarantee on one quality dimension. The guarantee consists of a *hasEffect*-Property which is connected to a quality-rule; in our example the rule guarantees a response time lower than 200. Each quality dimension is connected with a metric with the *hasMetric*-Property.
The example is continued in Figure 4. We are looking for semantic equivalent quality dimensions. In Figure 4 we present on the left side the specified quality

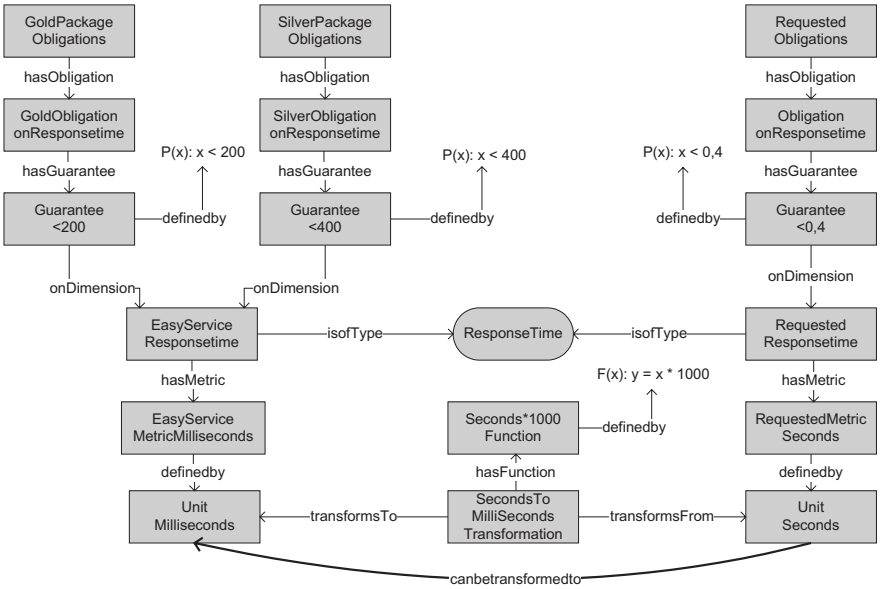

Fig. 4. Matchmaking of quality dimensions.

dimension *ResponseTime* of the gold and silver package and on the right side the requested quality dimension *RequestedResponsetime*. We have two quality dimension individuals of the same class but with different identifiers. To check if both parties mean the same quality dimension we use the *isofType*-Property. It is a binding-property to map to an external taxonomy which specifies the semantic relation of the quality dimensions. We see, that both specified dimensions are semantically related to the external specified individual *Responsetime* and therefore semantically equivalent.

Afterwards we check if both used metrics are semantically equivalent. Easy Service Company uses the individual *EasyServiceMetricMilliseconds* to measure its response time, which measure unit is milliseconds. Whereas the requested guarantee on response time is expressed in seconds, which is specified by the *definedby*-Property between the *RequestedMetricSeconds*- and the *UnitSeconds*-Individual.

At this point we use the Unit-Transformation-Ontology to specify the functional relation between units. Without this ontology we could only match quality dimension guarantees with the same measurement unit against each other, but we also want to match guarantees whose measurement units can be transformed into each other. If we have two *Unit*-Individuals, e.g. *UnitMilliseconds* and *UnitSeconds* which have a transformation between them, we specify a transitive *canbetranformedto*-property.

We can conclude that there is a stepwise transformation from seconds to mil-

liseconds by multiplying seconds with 1000. Functions are expressed by function groundings, which can be realized by a rule or an execution of a service. The function grounding is used to adapt the guarantee rules to equal units, e.g. transforming the rule $P(x) : x < 0,4$ to $P(x) : x < 400$. Afterwards we can match the guarantees. As both packages contribute the necessary quality level, the requestor takes the packages with the lowest price which is the silver package.

5 Related Work

Some work dealt with specifying quality of service and service level agreements. WSML [SDM02] and WSLA [ea02] are specification languages for SLA. Both languages allow specification of quality dimensions, metrics and guarantees, but both approaches lack the usage of semantics and therefore cannot bridge the gap of using different terms and semantic related metrics.
In OWL-S [ea04] QoS is specified as service parameters but lacks the specification of metrics and guarantees. It also lacks the specification of functional relations between metrics and is therefore not applicable for quality-aware service discovery. WSMO [RKL04] has indirect support for QoS by using non-functional properties for quality dimensions and functional properties for relation between quality dimensions. Their approach lacks semantic definition of stepwise transformation between metrics and therefore you have to specify a transformation for each unit into another unit.
DAML-QoS [ZCL05] is an ontology for QoS-Specification. Like our approach DAML-QoS differentiates between QoS-Offers and QoS-Requests but does not support the specification of service-packages and pricing. It uses object-oriented identifiers to bridge different terminologies and metrics in quality dimensions. These identifiers are predefined metrics. Our approach has an underlying Unit-Transformation-Ontology to define functional relations between metrics.

6 Conclusion

This paper introduces an ontology for quality aware service discovery. We have presented the necessary elements of quality aware service discovery and importance of integrating quality aspects in service integration. A description language needs flexibility for service level packages and service providing parties. It must also handle different terms in specifying QoS-Dimensions. Therefore we propose our ontology for semantic modelling of Service-Levels.
This is followed by an example for matchmaking of Service-Level-Offers and Service-Level-Requests 4. Our ontology is able to express the necessary generality for the specification of Service-Levels, Service-Level-Offers and Service-Level-Requests. Additionally we are able to bridge the usage of different terms by using binding-properties to include external ontologies.
Furthermore we are able to specify transformations between metrics. A transitive property is used to specify semantically if there exists a stepwise transformation

between two different metrics.
Currently we are testing a prototype matchmaker for different examples. We are confident to realize an automatic service brokering system and extend our approach not only on simple services but for whole service compositions.

References

[BHMS05] R. Berbner, O. Heckmann, Andreas M., and Ralf S. Eine dienstgüte unterstützende webservice-architectur für flexible geschäftsprozesse. *Wirtschaftsinformatik*, 47(4):268–277, 2005.

[ea02] A. Dan et al. Web service level agreement (wsla) language specification. In *In documentation for Web Services Toolkit, version 3.2.1.* International Business Machines Corporation (IBM), August 2002.

[ea04] D. Martin et al. *OWL-S, OWL-based Web Service Ontology*, 2004. URL: http://www.daml.org/services/owl-s/1.1/.

[IBM03] IBM. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, 2003. URL: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[JT04] M. C. Jaeger and S. Tang. Ranked matching for service descriptions using daml-s. In Janis Grundspenkis and Marite Kirikova, editors, *Proceedings of CAiSE'04 Workshops*, pages 217–228, Riga, Latvia, June 2004. Riga Technical University.

[MH04] E. Miller and J. Hendler. *Web Ontology Language (OWL).* World Wide Web Consortium (W3C), 2004. URL: http://www.w3.org/2004/OWL/.

[RKL04] D. Roman, U. Keller, and H. Lausen. Wsmo - web service modeling ontology. In *DERI Working Draft 14.* Digital Enterprise Research Institute (DERI), February 2004.

[RS04] J. Rao and X. Su. A survey of automated web service composition methods. In *Semantic Web Services ancd Web Process Composition, 2004, Springer-Verlag, pp 43-54.* First International Workshop, SWSWPC 6. July 2004, San Diego, July 2004.

[SDM02] A. Sahai, A. Durante, and V. Machiraju. *Towards Automated SLA Management for Web Services.* Hewlett-Packard Laboratories Palo Alto, July 2002. URL: http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf.

[Wor04] World Wide Web Consortium. *WS-CDL, Web Services Choreography Description Language*, December 2004. URL: http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

[ZCL05] C. Zhou, L.-T. Chia, and B.-S. Lee. Semantics in service discovery and qos measurement. *Wirtschaftsinformatik*, IT Pro(March/April):29–35, 2005.

Reusable Web Service Choreography and Orchestration Patterns[1]

Guadalupe Ortiz, Juan Hernández, Pedro J. Clemente

University of Extremadura
Quercus Software Engineering Group
http://quercuseg.unex.es
{gobellot, juanher, jclemente}@unex.es

Abstract. Web Service composition provides a new and successful way of enabling interoperability among different web applications. In this paper, an approach to the implementation and reuse of orchestration and choreographies' functional patterns by using aspect-oriented techniques is provided. The interaction patterns are implemented as abstract aspects, to specialize them in the contexts where they could appear, without the need of knowing or modifying the code of the services composed, thus improving modularity, scalability and flexibility in the compositions, whilst allowing reusability of the different patterns previously invoked.

1. Introduction

Web Services composition has become the new way to compose various applications through the Web, originating the appearance of the two new terms *orchestration* and *choreography* [1]. They refer to two different ways of managing business connectivity, and have arisen in a moment in which many companies have begun to incorporate Web Services to their deployments.

Many languages have been proposed and discussed for Web Services composition, but among them we can only stress two within Web Services: BPEL4WS [2] and WS-CDL [3]. To start with, BPEL4WS (*Business Process Execution Language for Web Services*) allows users to describe the control logic for coordinating different Web Services, which takes place in a process flow; it is mainly focused on permitting orchestrations to be defined. Alternatively, WSCDL (*Web Service Choreography Description Language*) specification is the standard proposal from the W3C, mainly centered on choreographies. In any case, the boundaries between choreographies and orchestrations are not very clearly defined and therefore we cannot establish limits for the given approaches' application.

Let us consider we have already defined a composition, orchestration or choreography, and we find the same functional pattern in another context. It would be desirable to be able to reuse the pattern and simply adapt it to the specific context, not

[1] This work has been developed with the support of CICYT under contract TIC2002-04309-C02-01.

to have to rewrite the composition code. Furthermore, we do not want all the composition code scattered and highly coupled to the main application, but to have different patterns in a modularized way so as to specialize them should they be necessary. In spite of its being influential to both evolution and maintenance in Web Services, the market has not proposed a good answer to this matter for the time being, as the previously mentioned standards do not allow to define independent patterns in order to reuse them in a later stage. For this reason, we propose composing Web Services by using *Aspect-Oriented Programming* (AOP) [4], thus totally decoupling the various Web Services composed and facilitating service maintenance and modularization, as well as reusability of their interaction patterns. In this paper we shall centre mainly on how to apply AOP techniques to Web Service composition and reuse the interaction logic of the previously defined composition.

The rest of the paper will be arranged as follows: a case study is presented in Section 2 to identify the aforementioned problems and illustrate the orchestration and choreography concepts. Section 3 underlines the shortcomings of the case study implementation using different kinds of tools. Section 4 moves on to outline the way in which AOP can help to solve these problems. Section 5 shows how *AspectJ* has been used for implementing composition interaction patterns, whereas Section 6 shows how to specialize these patterns in a specific context. To finish with, related approaches are considered in Section 8, and discussion and conclusions are presented in Section 9.

2. Web Service Choreographies and Orchestrations: our Case Study

As we previously mentioned, the terms *choreography* and *orchestration* are not clearly discriminated and, depending on the author, they are used with the same or different meanings. In our view, choreographies refer to the sequence of messages that may be interchanged among various Web Services, wherein each one describes its role in the communication and none of the services involved but an external layer controls the process,. On the contrary, in the case of orchestrations, one of the services involved in the composition is the one that manages all the interaction logic.

As current Web Services are not prepared to be choreographed and we are not supposed to be able to modify services' code, we suggest the use of an auxiliary service to lead the choreography interaction logic (an alternative when the service code can be modified can be found in [5]). In this respect, the composition implementation using aspect-oriented programming is illustrated in both types of composition with a common case study, which is shown in *Figure 1*.

Consider that we have a travel agent service offering products as booking a flight or a hotel. We may be interested in offering a new product which consists of renting a car with a driver to go from one city to another within the United States, which would be the operation invoked by the client, *carRenting*. The cost of the service will depend on the distance between the two cities. In order to offer the new product we invoke the operation *getDistance* in *distanceService* and each mile is multiplied by one dollar. The client will then be able to choose the currency for payment, so the

convert operation will be invoked in the *currencyConverterService*. Finally, the cost will be paid by credit card, which will be checked by invoking the *creditCardService validateNumber* operation**.** Once this is finished, the client receives the final result.

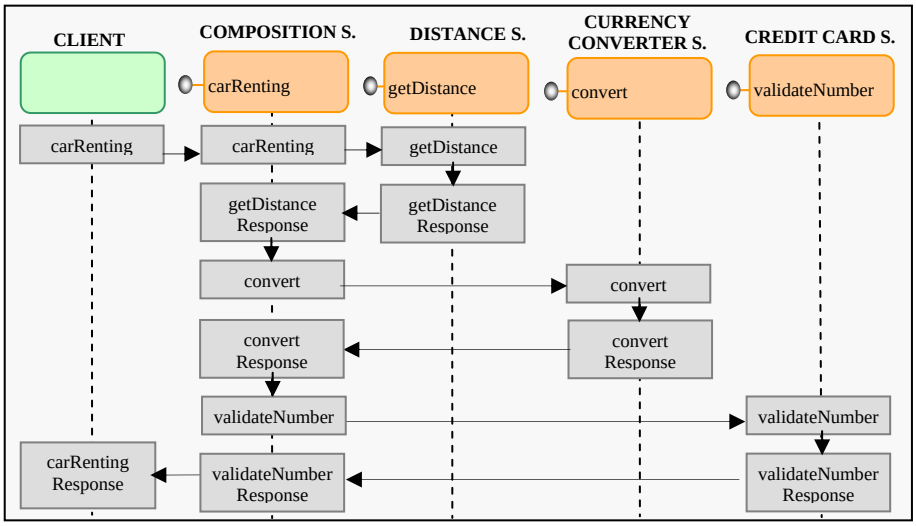

Fig. 1. Example of an invocation order in a composition

In this case, we would be talking about an orchestration, as our travel agent service manages the interaction logic to offer this new product. For instance, in *Figure 1*, the composition service would be the travel agent service, where we are only showing the behaviour of one of its operations. If we did not have a travel agent or any other service to incorporate the new product, but we wished this kind of composition to be offered, we would be talking about a choreography, where the different services would be composed but none of them would manage the interaction logic, and so we should have to create an auxiliary service, whose only purpose would be to offer this choreography. The composition service in *Figure 1* would be this auxiliary service.

3. Composition Implementations

In this section we are going to analyse the advantages and disadvantages of the composition implementation defined in *Figure 1* with two different tools (briefly described due to space restrictions). First, we will use a general purpose tool and then we will utilize a tool based on the creation of an XML file that leads the composition.

The first option could be using the Sun tool JWSDP. When we implement the composition with this tool, both in the case of the choreography and the orchestration, this implementation does not perform many of the basic mainstay of programming, as we will briefly go on to argue:

- **Modularity**: The code related to stub creation and services invocations related to the specific platform used is very strongly coupled in the module, which creates hard dependences between the composition and the particular platform used.

- **Reusability**: If we wanted to reuse an interaction pattern modelled by the composition we would have no chance to do so, as all the code related to the services composed is mixed with the one which models the interaction logic.
- **Maintenance and evolution**: The maintenance of the application is hindered as we would have to modify our main applications when any change was made in the remote services. This would result in time wasted due to bad design and therefore damage in evolution.

To exemplify the second option we have chosen BPEL4WS, as WS-CDL is not so established, but it is enough to show both types of composition shortcomings. If we examine the same points we previously analysed for the general purpose tool implementation, we will find that we have improved as far as modularity, maintenance and evolution are concerned but we still find some essential aspects not desirable in an application. Regarding maintenance and evolution, the only drawback is the need to generate the XML file again when the remote services change. Concerning *reusability*, we can only reuse the orchestration completely, as a new service, provided we have exactly the same requirements in another application, in which case we would invoke the operations offered by the orchestration. Nevertheless, we still have no chance to reuse the interaction pattern in other contexts.

4. AOP and Compositions

AOP arises because of the problems detected in *Object-Oriented Programming* (OOP). OOP is supposed to permit the encapsulation and modularity of related data and methods which address a common goal, but this is not always possible. We could simply have transversal concerns, which cannot be included in the logical structure of the code by functionality. As a result of these concerns, code is scattered and tangled all over our application. AOP describes five kinds of elements to modularize these crosscutting concerns: firstly, we have to define the *join point* model which indicates the points where we could include new behaviours. Secondly, we have to define a way to indicate the specific *join points* to specify in which points of the implementation we wish to insert the new code. Next, we ought to determine how to specify the new behaviour to be included in the *join point* referred to. We would then encapsulate the join points and their corresponding behaviours into independent units. Finally, a method to weave the new code with the original one has to be applied [6].

In any composition a certain order has to be followed when making invocations: thus, sometimes we will have services which have to be invoked before others, those which will have to be invoked after others have finished and finally, services which may or may not be invoked depending on the result of other invocations. AOP allows us to model these interactions into independent units called *aspects*, maintaining the independence among the services composed. As we will see in the next section, while modelling our composition using these techniques, we can reuse common functional patterns in different contexts in which they could appear, thus improving our applications. AOP is also successfully used within the Web Service domain for implementing non-functional properties at compilation time, as *logging* or *timing*, in a modularized and decoupled way [7].

5. Implementing Functional Patterns with AOP

When the developer considers the possibility of implementing a pattern that can be reused later in different contexts, he has to decide on the grain of the pattern. In our example we may adapt to a coarse grain pattern (*Figure 2a)*), or we could also have very fine grain one, as shown in *Figure 2b)*. Furthermore, we can have a pool of typical functional patterns with different grain to be implemented as a composition of services, as, for instance, a purchase with a previous credit and stock check or related pursuits based on the result of previous searches, or our case study pattern…, so patterns can be reused in the different contexts in which they may appear.

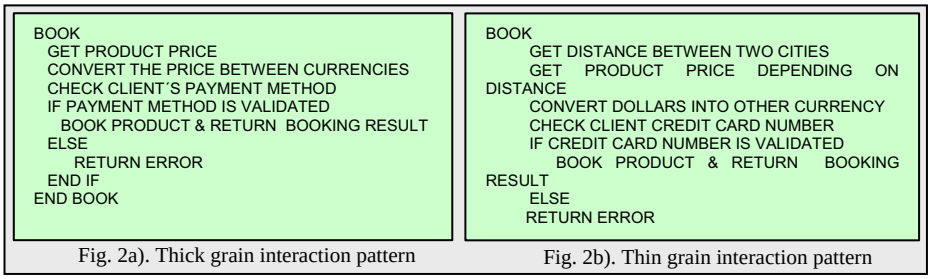

Fig. 2a). Thick grain interaction pattern

Fig. 2b). Thin grain interaction pattern

Fig. 2. Coarse and fine grain interaction patterns

By using AOP, interaction patterns can be implemented as abstract aspects: we may define an abstract aspect which describes the interaction pattern for a specific kind of composition. Both choreography and orchestration have the same abstract aspect implementing the interaction logic. The difference, as we mentioned before, is:

- The interaction logic is linked to a particular method of a specific service which takes part in the composition in the case of an orchestration.
- We have to create an auxiliary service with an auxiliary method, which is linked to the interaction logic for the choreography to be available.

To illustrate this point, we have represented the pattern in *Figure 2b)* as an AspectJ aspect in *Figure 3a)*. This aspect is integrated by the abstract *pointcut Renting*, which refers to a general renting operation. As it is an abstract aspect, we still do not have to specify in which method execution the new code is to be inserted. In the *advice* linked to *pointcut Renting*, the arguments are obtained from the context and used for the invocation of the abstract classes that represent the remote methods. To implement a car rental operation, as our example goes, we first have to invoke *getDistance*, then *getCurrencyConverter* and, finally, *checkCreditCard*, all of them abstract. We do not need to instruct on the specific services to be invoked nor their operations, since this will be done when we create the specific aspect that inherits the abstract one.

6. Specializing and Reusing Patterns with AOP

In the very moment we need to implement a specific composition of the case study pattern, we will inherit the composition pattern from the abstract aspect, as we can see

in *Figure 3b)*. Hence, if we want to implement our specific composition case study using the designed pattern, we have to implement the aspect which extends the abstract one, specifying the particular stubs of the services involved. We also have to redefine the abstract operations, *getDistance*, *getCurrencyConverter* and *checkCreditCard*, now to implement their actual behaviour, invoking the corresponding operation in the remote services. Finally, pointcut *Renting* has to be redefined, indicating the point in which the new code is to be inserted. In the case of orchestrations, this point is the operation offered in the *orchestrator* service for offering this new product. With choreographies, a new service has to be created which will provide the operation offered by the choreography in question, namely the point intercepted by the pointcut.

```
abstract aspect OrchesAbstractAspect{

public abstract double getTheDistance(Object[] args);
public abstract float getCurrencyConverter( Object[]
args);
public abstract String checkTheCreditCard( Object[]
args);

abstract pointcut Renting();

String around () : Renting(){
 Object[] args =thisJoinPoint.getArgs();
 Try{
  double DistanceResult= getTheDistance(args);
       if (DistanceResult==-1)
   return ("Distance Service not found");
  else{
   args[args.length()+1]=DistanceResult;
   float ConverterResult=getCurrencyConverter( args);
   if (ConverterResult==-1)
    return ("Currency Converter Service not found");
   else{
    args[args.length()+1]=DistanceResult;
   String CreditCardResult = checkTheCreditCard(args,);
    if (CreditCardResult.compareTo("-1")==0)
      return ("Credit Card Service not found");
    else  if (CreditCardResult.compareTo("-2")==0)
      return ("Invalid Card. Amount:"+ ConverterResult;)
    else
      return ("Valid Card. Amount"+ ConverterResult);
 } } } } catch (Exception ex) {    ex.printStackTrace();    }}
```

Fig. 3a). Abstract aspect for pattern in Figure 4b)

```
public aspect CarRentingAspect extends
OrchesAbstractAspect {
//code related to the distance stubs creation
//code related to the currencyConverter stubs creation
//code related to the creditCard stubs creation
pointcut Renting (): execution(public * *.carRenting(..));

public double getTheDistance(Object[] args){
  double distanceResult = -1;
  try{
   distanceResult= distance.getDistance(args[1], args[2]);
  }catch(Exception e){e.printStackTrace();}
  return distanceResult;}

public float getCurrencyConverter(Object[] args){
  float currencyConverterResult=-2;
  try{
        currencyConverterResult=
currencyConverter.convert
                ((float)args[5], "USD",(String)args[4]) ;
  }catch(Exception e){e.printStackTrace();}
  return  currencyConverterResult;}

public String checkTheCreditCard(Object[] args){
  String creditCardResult="-1";
  try[
      creditResult= creditCard.validateNumber(args[3]);
      if (creditResult==false)   creditCardResult="-2";
      else                       creditCardResult="1"
  }catch(Exception e){e.printStackTrace();}
  return creditCardResult;}
```

Fig. 3b). Inherited aspect for the case study

Fig. 3. Abstract and inherited aspects for pattern in Figure 2b)

To sum up, we have a representation of our final proposal in *Figure 4*, where we have depicted in bold the abstract interaction pattern, which is the main reusable element. This pattern could be selected from a pool of available patterns. In this respect, whenever we want to use the given pattern, we only have to specify the particular operation our choreography is going to offer or determine the operation which is going to be used for the orchestration in the main service and extend the pattern with the specific parameters and operation specifications, both modules being represented in italics in the said Figure. In this specification we are already linking the pattern with the remote services involved in the composition, which are the only elements whose code is not needed for implementing the composition.

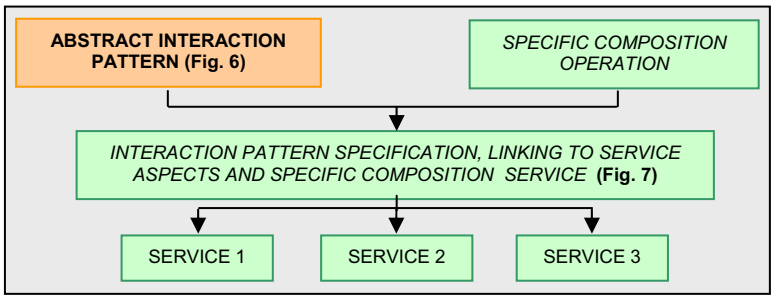

Fig. 4. Diagram of composition pattern reuse with AOP

7. Related work

Web Service composition is a very common research area nowadays, and plenty of studies can be mentioned on this sphere. Although there are undoubtedly important infrastructural issues in this field, there seems to be little discussion in the specialized media on how to use AOP techniques for this functionality.

First of all, the idea of encapsulating the composition logic can be examined in various articles. B. Orriëns et al. propose a packaging mechanism, named *web component*, for developing applications by combining various existing Web Services [8]. In contrast to our proposal where only one additional language is needed, they propound various different stages with their different specification languages. Otherwise, L. Melloul et al. use high level patterns to abstract composition functionality with their own framework [9]; however, especially in orchestrations, they do not decouple the pattern from the main application. Besides, C. Zirpins et al. propose another mechanism to address different coordination alternatives for Web Service compositions [10], but mainly focus on the use of coordination policies and idioms, far from our aspect-oriented proposal.

On the other hand, in the area of aspect-oriented techniques with Web Services we specially distinguish a paper focused on a new language based on XML, AO4BPPEL, an aspect-oriented extension for BPEL [11]. A. Charfi *et al.* use this new language for implementing business rules in BPEL compositions. In contrast with our proposal based on languages with wide application support, they need a new weaver for the proposed language, not available on the Web nowadays. Finally, M.A. Verheecke et al. suggest the use of a dynamic aspect-oriented language called JAsCo for decoupling services from the application which invokes them, focusing mainly on the client side [12]. In contrast, our proposal uses a general use aspect-oriented language and is not centred on the client side.

7. Discussion and Conclusions

The results obtained in this study show how aspect-oriented programming is really useful in Web Service composition. Aspect-oriented techniques have been used for implementing and reusing Web Services composition interaction patterns, and they have proved to improve their modularity, scalability and flexibility.

In particular, we have used AspectJ, a very well-known aspect-oriented language, to illustrate our proposal, and it has provided a good way to modularize and encapsulate the message exchange, decoupling the services among them and offering an easy maintenance of the application.
Regarding modularity, all the composition-related code in the aspect is now found completely modularized, decoupling the remote services entirely from the implemented application and thus having no trace of scattered and tangled code. As far as reusability is concerned, it is important to emphasise the possibility of defining abstract aspects which describe different grain functional patterns. These abstract aspects can be later inherited from a particular aspect which only has to particularize the arguments and the specific operations to be invoked in order to implement the desired composition. Hence, we can have a pool of abstract interaction patterns and select one of them, should they be necessary in a specific context, which implies a very important benefit in this field. Finally, we can assert that maintenance and evolution are more convenient, since we do not have to modify the main application code at all for changing the services composed, but only the modularized aspect. In this sense, our application will be more reliable.

8. References

[1] Peltz, C. Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS. Web Services Journal, July 2003.
[2] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leimann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. Business Process Execution Language for Web Services Version 1.1. Microsoft, May 2003.
[3] Kavantzas, N. Web Service Choreography Description Language (WS-CDL) 1.0 Editors draft. W3C, December 2004.
[4] Kiczales, G. Aspect-Oriented Programming, ECOOP'97 Conference proceedings, Jyväskylä, Finland, June 1997.
[5] Ortiz, G., Hernández, J., Clemente, P. J.Preparando los servicios web para las coreografías Proc.I Jornadas Científico-Técnicas en Servicios Web at Congreso Español de Informática, Granada, Spain, September 2005
[6] Elrad, T., Aksit, M., Kitzales, G., Lieberherr, K., Ossher, H.: Discussing Aspects of AOP. Communications of the ACM, Vol.44, No. 10, October 2001.
[7] Ortiz, G., Hernández, J., Clemente, P. J. How to Deal with Non-Functional Properties in Web Service Development, Proc. of the International Conference on Web Engineering, Sydney, Australia, July 2005
[8] Orriëns, B., Yang, J., Papazoglou, M.P. A Framework for Business Rule Driven Web Service Composition. Workshop ER Conference, Chicago, Illinois, October 2003
[9] Melloul, L., Foz, A. Reusable Functional Composition Patterns for Web Services. Proc. Second International Conference on Web Services (ICWS04), San Diego, CA, July, 2004.
[10] Zirpins, C., Lamersdorf,W., Baier, T. .Flexible Coordination of Service Interaction Patterns. Proceedings at ICSOC04, 2nd Int. Conference on Service Oriented Computing, New York City, USA
[11] Charfi, A., Mezini, M.. Aspect-Oriented Web Service Composition, Proc. 2004 European Conference on Web Services (ECOWS 2004), Erfurt, Germany, September, 2004.
[12] Verheecke, B., Cibrán, M..A. AOP for Dynamic Configuration and Management of Web Services. Procedings International Conference on Web Services, (ICWS-Europe'03) Erfurt, Germany, September 2003

Engineering Distributed Service Compositions

Thomas Cottenier[1, 2], Tzilla Elrad[1]

[1] Concurrent Programming Research Group, Illinois Institute of Technology
3300 S Federal Street, 60616 Chicago, IL, USA
{cotttho, elrad}@iit.edu
[2] Software and Systems Engineering Research, Motorola Labs
1300 E Algonquin Road, 60173 Schaumburg, IL, USA
thomas.cottenier@motorola.com

Abstract. Current technologies have difficulties supporting distributed service compositions, especially transient distributed compositions. This paper discusses technologies and engineering practices that would enable service collaborations to be deployed, refined and customized on-demand, by third parties, without compromising the safety and the integrity of the service and processes. Processes that are exposed for late distributed composition need to expose more information about themselves than processes designed for simple consumption. Aspect-Sensitive Processes expose a description of their invocation steps. The actions exposed by this description can be refined by third parties, as long as the resulting executable processes do not violate the invariants of the process, as expressed in behavioral interface descriptions. The framework provides a ground for experimentation with dynamic and distributed workflows, and proposes a common specification language for defining advanced service composition schemes, such as adaptive choreographies, mobile agents and distributed aspect-oriented compositions in web service environments.

1 Introduction

Different service composition mechanisms reflect different patterns of mutual relationships between organizations.

Service composition languages such as the Business Process Execution Language (BPEL) [8] are centralized *orchestration* languages. A BPEL engine controls all message exchanges between an organization and its partners. It is therefore suited to implement master-slave *value chain* relationships, between a dominant organization and supply and distribution partners.

As organizations further specialize, and relationships are established between partners that participate in other value chains, more decentralized collaborations develop between organizations. They form *value networks*, where organizations relate with each other in a more peer-to-peer manner. Each value chain might have its own specificities. It might require from the partners that they support a specific security or transactional integrity protocol, or support a particular conversation protocol. Therefore, partners are pressed to implement processes that adapt the services they provide to the specificities of each value chain. Within a value network, a value chain involves several coordinated processes. The value chain does not have a single center of

control anymore. We say that the service composition is decentralized [1]. Decentralized compositions are implemented by coordinating the BPEL processes of the participants. A *choreography* description is a specification of how these processes are coordinated. It specifies the conversations that take place between partners.

The emergence of more dynamic value networks, such as *trading communities,* forces the returned value of the relationships to be obtained over a small amount of interactions, with short term partners. The benefit of participating in virtual communities lies in the large number of potential interactions, and the variety of available services and resources. Yet, partners need to accommodate a very loosely coupled and dynamic environment. Participants need the ability to quickly establish new relationships with peers. They need the ability to mutually adapt their processes to make conversations compatible, in a competitive timeframe.

Current service composition mechanisms are not suited for this kind of environments. For now, distributed service compositions can only be established through explicit consultation between the participating parties. Distributed compositions are therefore only deployed between long term partners, for specific cooperative business applications.

This paper discusses technologies and engineering practices that would enable distributed compositions to be deployed, refined and customized on-demand, by third parties.

2 On-Demand Choreography Deployment

2.1 Traditional Choreography Deployment

Distributed composite services can only be deployed after an offline agreement between the collaboration partners is reached.

Within a distributed composition, we can identify two distinct business partner roles: one initiator party and one or more reactive parties. The initiator party is the partner that gets the most value out of the composition. This party publishes the endpoints of the composite service. Typically, reactive parties deploy custom processes at the request of the initiator party, in order to satisfy his requirements.

At the request of the initiator party, domain experts of the different domains meet, and negotiate the conversations to be supported by their processes. These agreements are described in a choreography model, expressed in a language such as the Choreography Description Language (WS-CDL) [6] [7]. A *choreography model* describes the observable sequence of message exchanges between the peers that participate in the collaboration, viewed from a global perspective. The responsibilities of each party are defined by a *behavioral interface model*. It describes the view of the choreography description from the perspective of a single partner. It complements the structural interface of the services (WSDL) by defining dependencies between the different interactions of the collaboration. It also specifies the policies (security, transactional dependencies, etc) and the responsibilities of the party in the service collaboration.

Each party then implements the individual orchestration processes that conform to the choreography agreement. An *orchestration model* details the communication

actions and the internal actions in which a service engages to implement its behavioral interface. Internal actions include invocations of internal software modules and data transformations. Orchestrations are executable by an engine, and are expressed in an orchestration language such as the BPEL [8].

Behavioral interfaces can be automatically generated from a choreography description, and orchestration skeletons can be generated from behavioral interfaces [6]. In the next sections of the paper, it is also assumed that it is possible to check whether the orchestration of an existing process conforms to a behavioral interface. It is therefore possible to detect situations where an orchestration does not meet the requirements that are expected by its partners, or whether a process violates essential invariants.

2.2 Inter-Process Coupling

The traditional approach to choreography deployment introduces coupling between the participant processes. We identify two different kinds of inter-process coupling:

- *Conversational coupling*: The business logic conversations couple the processes deployed by the reactive parties tightly to the processes of the initiator party. They are highly specialized for use in a specific context, and are very unlikely to be useable outside the scope of the collaboration.
- *Protocol coupling*: The collaboration commits to a specific security or transactional integrity protocol such as WS-Trust or WS-Transaction. Many of these specifications are not standardized and are still evolving. The logic that integrates these concerns within the collaboration is tangled with the business logic of the processes [2]. Tangling impedes the maintainability of the processes.

2.3 On-Demand Choreography Deployment

Conversational coupling can be addressed by enabling the initiator party to choreograph existing reactive processes to build a composite service that fits its requirements. The reactive parties do not have to deploy a custom executable process for each of the collaborations they participate in. Rather, they deploy more generic virtual processes, which can be refined by the initiator into more specific executable processes. Conversational coupling is thereby reduced, because the reactive parties are oblivious to the more specific conversations deployed by the initiator.

Protocol coupling can be addressed by decomposing the service collaboration into a set of collaboration layers [3]. Each collaboration layer encapsulates a specific concern within the application, such as security or transactional integrity. The implementation of these layers is defined with respect of the business logic of the application. This decomposition allows the initiator to swap the protocols being used when needed, and enables the reactive parties to be oblivious to the non-functional requirements of the initiator. Figure 1 depicts the decomposition of a distributed application into a business logic layer and a layer that handles the coordination of the application

with CAF [9]. Layered decomposition proceeds by transparently hooking the roles of each party within the protocol at the right locations in their executable processes.

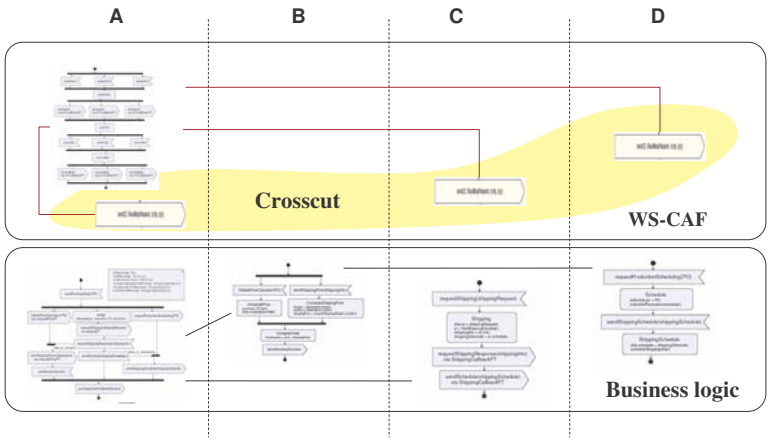

Fig. 1. Decomposition of a distributed composition into distinct collaboration layers

Both process refinement and layered decomposition require the initiator to have the ability to inspect and refine the processes exposed by the reactive parties. Process inspection requires part of the reactive processes to be visible to the outside world. Process refinement necessitates the ability to deploy data mapping operations into the processes and the ability to refine the default control and data flow of the process.

The data mappings to be performed can be derived from the structural interfaces of the service endpoints. They can therefore be generated semi-automatically from the WSDL's of the participating services in an open standard format, such as XSL.

Finally**,** the executable processes resulting from the process compositions and refinements should not compromise safety, and should not violate the essential properties of the processes defined on the host. It should therefore be possible to validate the executable processes generated from the compositions against specifications that constraint the behavior of the processes.

3 Executable Choreography Framework

The Executable Choreography Framework (ECF) [4][5] introduces a language to specify process refinements and a platform extension to enable runtime refinement and composition of processes. Process refinements operate on Aspect-Sensitive Processes (ASP's). An ASP provides an abstract description of a reactive process and is associated to a set of behavioral interfaces, which constrain the executable processes that can be composed from the ASP. Figure 2 depicts the different components of the ECF.

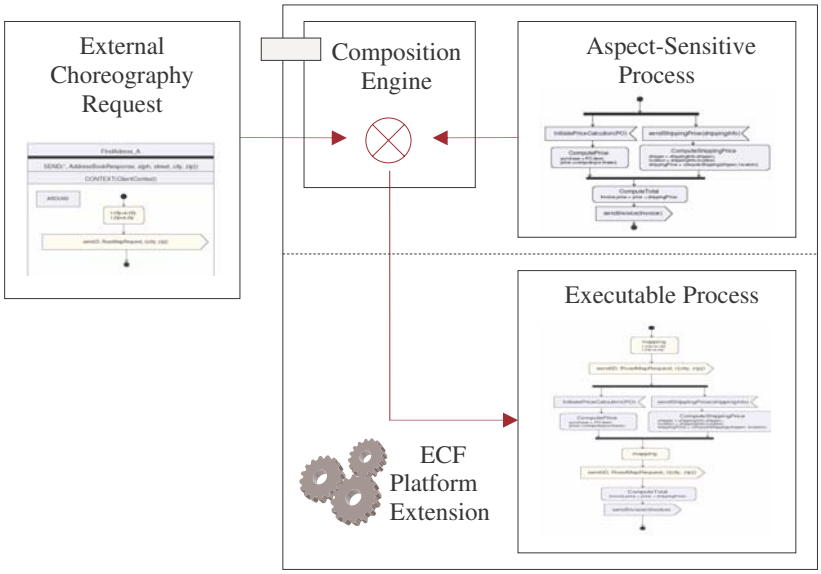

Fig. 2. Components of the Executable Choreography Framework

3.1 Behavioral Constraints

Constraints on the process composition are expressed as behavioral interfaces. These interfaces define the constraints (control-flow dependencies, data-flow dependencies, message correlations, time constraints, transactional dependencies, etc.) that must, may or must not be implemented by the processes composed from the ASP description. These can be derived from existing choreography descriptions, or specifications such as the Composite Application Framework (WS-CAF) [9] or WS-Trust, by mapping the overall collaborations to the roles assumed by the host. These behavioral interfaces specify what is required from the processes executing on the host. All processes composed by third parties need to conform to these specifications. We assume that the processes that are composed on the domain can be checked for correctness, with respect to the behavioral interfaces defined on the domain. The formal specification of the behavioral constraints on the composition is an area of further research.

3.2 Aspect-Sensitive Processes

ASP's expose a process description that conforms to the behavioral interface. Process descriptions are not executable processes. They describe portions of executable processes, by describing parts of their control and data flow. The internal implementation details of the executable processes do not have to be revealed. The process description makes public the steps that may need to be refined in order for it to participate in

new collaborations. All steps of the process description are hooks where new actions can be introduced. For example, the default process description is composed of three sequential actions, the reception of a request message, followed by the internal invocation of the service and the emission of the service response back to the requester. A third party can refine this process, by introducing additional elements of composition logic, before, after or around these actions.

Each intermediary step of the ASP description needs to be annotated with a WSDL definition of the type of the data that is expected at that point. This enables third parties to refine the process descriptions and adapt the data transformation steps as to ensure the type correctness of the process.

Process descriptions can be refined and composed by third parties, as long as the resulting executable processes do not violate the behavioral interface of the ASP. Processes that are exposed for later composition by third parties need to expose more information about themselves than processes designed for simple consumption. They need to expose a description of their invocation process, and the structural interfaces (WSDL's) of their internal processing steps.

3.3 Executable Choreography Language

The Executable Choreography Language (ECL) is a XML-based language to define refinements on the default control flow of service invocation and execution. Given a choreography description, either provided as an activity diagram or a WS-CDL specification, the ECF partitions the distributed workflow into ECL rules. ECL rules specify a set of actions to be performed when a message of interest is intercepted within a target process. For clarity purposes, XML ECL rules are represented in a graphical notation, as shown in Figure 3. It is composed of an activity context declaration, an event pattern, a set of conditions and an action:

1. CONTEXT: An Activity Context declaration uniquely identifies the rule. At runtime, the context encapsulates information that relates to a distributed activity. Activity contexts are propagated transparently from node to node, along the interactions of an activity.
2. EVENT: An Event expression defines when the rule should be applied. Any action exposed by the ASP can be intercepted. See the ACTION section for the list actions that can be intercepted. An event expression defines a pattern on the signature of the messages to be intercepted at that point.
3. CONDITION: The condition clauses identify the activities for which the refinement of the control flow should be applied. Interactions of an activity are discriminated based on their activity context information. See section 3.4.
4. MODIFIER: A modifier specifies when the rule behavior should be applied. The rule actions can execute either Before, Around (Instead) or After the events captured by the event pattern.
5. ACTION: The ECF takes control over the thread of execution of the service invocation or execution. The actions authorized by the ECF are:
   - Compound Action: Actions can be composed into named compound actions, using the sequence, fork, join, decision and merge control flow operators.
   - Accept Request/Response Action: wait for a request or a response message.

- Send Request/Response Message Action: send a service request or response.
- Invoke Service Action: invoke the functionality of a local resource
- Set Timer Action, Reset Timer Action and Accept Time Event Action: timing actions allow ECL rules to define timeouts and handle faults.
- Data Mapping Action: The SOAP messages intercepted by the ECF can be transformed according to a XSL specification. SOAP messages can be aggregated and data can be consolidated.

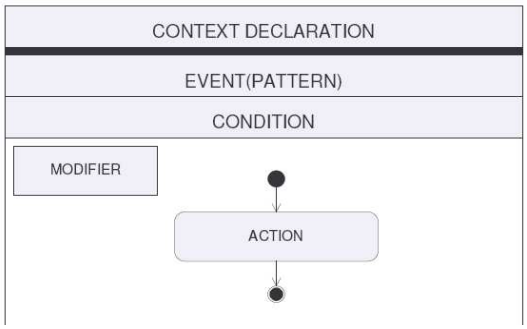

Fig. 3. Graphical representation of an ECL rule

3.4 Executable Choreography Platform Extension

ECF-enabled platforms can interpret ECL rules, and deploy the corresponding control and data flow logic accordingly, in the native language of the platform. The ECF platform extension implements 3 distinct functionalities: message interception, transparent activity context propagation and dynamic rule deployment.

- **Message Interception:** Messages flowing in and out the container can be intercepted within the process, at any point that is exposed by the ASP description. When a message matches the event pattern of an ECL rule within its activity context, the ECF platform extension takes control over the thread of the service request or response, and injects the rule behavior, before, after or instead of the intercepted event.
- **Activity Context Propagation:** The ECF provides transparent context propagation within a distributed activity. Activity contexts are piggybacked in the headers of the intercepted messages. The ECF platform extension ensures that contexts are propagated from node to node, along the interactions of a same distributed activity. Context propagation helps managing the life-cycle of distributed activities. The ECF concept of activity context is derived from the Composite Application Framework (WS-CAF) specification [9].
- **Dynamic Deployment:** ECL rules can be deployed on remote containers by third parties. ECF-enabled platforms expose the endpoint of the composition engine as a Web Service. The context identification of the ECL rule ensures that only activities that are initiated by the initiator party can be intercepted by the rule.

4 Conclusions

This paper discusses techniques to enable service collaborations to be deployed, refined and customized on-demand, by third parties, without compromising the safety and the integrity of the services and processes. Processes that are exposed for late distributed composition by third parties need to expose more information about themselves than processes designed for simple consumption. They need to expose a description of their execution process and a behavioral interface which constraints the executable processes that result from third party composition. The framework provides a ground for experimentation with dynamic and distributed workflows, and proposes a common specification language for defining advanced service composition schemes, such as adaptive choreographies, mobile agents and distributed aspect-oriented compositions in web service environments [5].

Acknowledgment

This work is partially supported by CISE NSF grant No. 0137743.

References

1. Chafle, G., Chandra, S., Mann, V., Nanda, M. G.: Decentralized Orchestration of Composite Web Services. Proceedings of the Thirteenth International World Wide Web Conference, New York, NY, USA, ACM Press (2004)
2. Charfi, A., Mezini, M.: Aspect-Oriented Web Service Composition with AO4BPEL. Proceedings of the European Conference on Web Services, Erfurt, Germany, LNCS 3250 (2004)
3. VanHilst, M., Notkin, D.: Using Role Components to Implement Collaboration-Based Designs. In Proceedings of the 11th ACM conference on Object-Oriented Programming, Systems, Languages, and Applications, San Jose, California, United States, ACM Press. (1996)
4. Cottenier, T., Elrad T, Prunicki, A.: Contextual Aspect-Sensitive Services, formal demonstration presented at the 4th International conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA (2005)
5. Cottenier, T., Elrad, T.: Executable Choreography Framework, technical report and formal demonstration to be presented at the 3rd International Conference on Service-Oriented Computing (ICSOC'05), Amsterdam, The Netherlands (2005)
6. Barros A., Dumas M., Oaks P., A Critical Overview of the Web Services Choreography Description Language (WS-CDL), BPTrends (2005)
7. Web Services Choreography Description Language (WS-CDL) Version 1.0, W3C Working Draft 17, http://www.w3.org/TR/ws-cdl-10/ (2004)
8. Business Process Execution Language for Web Services, BEA, IBM, Microsoft, SAP and Siebel Systems, http://www-128.ibm.com/developerworks/library/ws-bpel (2004)
9. Web Services Composite Application Framework (WS-CAF). Arjuna Technologies Ltd., Fujitsu Limited, IONA Technologies Ltd., Oracle Corporation, and Sun Microsystems, Inc, http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf (2005)

Modelling and Analysis of Time-related Properties in Web Service Compositions

Raman Kazhamiakin[1], Paritosh Pandya[2], and Marco Pistore[1]

[1] DIT, University of Trento
via Sommarive 14, 38050, Trento, Italy
{raman,pistore}@dit.unitn.it
[2] Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba, Mumbai 400 005, India
pandya@tifr.res.in

Abstract. In this paper we present an approach for modelling and analyzing time-related properties of Web service compositions defined as a set of BPEL4WS processes. We introduce a formalism, called *Web Service Timed State Transition Systems (WSTTS)*, to capture the timed behavior of the composite web services. We also exploit an interval temporal logic to express complex timed assumptions and requirements on the system's behavior. Building on this formalization, we provide tools and techniques for model checking BPEL4WS compositions against time-related requirements. We perform a preliminary experimental evaluation of our approach and tools with the help of the e-Government case study.

1 Introduction

Web services provide the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols [1]. Service composition [2] is one of the most promising ideas underlying Web services: new functionalities can be defined and implemented by combining and interacting with pre-existing services. Different standards and languages have been proposed to develop Web service compositions. Business Process Execution Language for Web Services (BPEL for short) [3] is one of the emerging standards for describing a key aspect for the composition of Web services: the behavior of the service.

BPEL opens up the possibility of applying a range of formal techniques to the verification of the behavior of Web services, and different approaches have been defined for verifying BPEL [4–7, 13]. We are interested in particular in those techniques that are applied to the verification of BPEL compositions: in this case, we have to verify the behaviors generated by the interactions of a set of BPEL processes, each specifying the workflow and the protocol of one of the services participating to the composition. Correctness of these compositions requires not only the satisfaction of qualitative requirements (e.g. deadlock freeness), but also of quantitative properties, such as time, performance, and resource consumption.

Time-related properties are particularly relevant in this setting. Indeed, in many scenarios we expect that a composition satisfies some global timing constraints, which can be satisfied only if all the participating services are committed to respect their own local timing constraints. Consider for instance an e-government scenario, where the distributed business process requires the composition of information systems and functionalities provided by different departments or organizations. The composite service can comply with the timing commitments w.r.t. the state regulations (e.g., the duration of document analysis phase) only if they are consistent with the time required by the participants to carry out their part of the process.

In this paper we present an approach for modelling and analyzing time properties of WS compositions defined by a set of BPEL processes. We want to stress the fact that the time properties we want to model and analyze are those that are critical from the business logic point, i.e., they refer to the time required by the participants to carry out their tasks and take their decisions, and to the assumptions and constraints on these times that guarantee a successful execution of the composition scenario. In e-government scenarios, their times are measured in hours and in days. The "technical" times, which are required, for instance, by the communications among BPEL processes and by the BPEL engines to manage incoming and outgoing message, are orders of magnitude smaller (seconds if not milliseconds) and can be neglected in the scenarios above.

This work is based on previous results on the untimed verification of BPEL compositions [8]. In that framework, implemented as a part of the Astro project toolkit (`http://www.astroproject.org`), the BPEL processes are encoded as State Transitions Systems (STS), and then their composition is verified using NuSMV model checker. In this paper, we define the formalism of *Web Services Timed Transition Systems (WSTTS)*, extending STS to allow for modelling the timed behavior of a composition. WSTTS are closely related to timed automata but incorporate design decisions and features consistent with the Web service composition. We also demonstrate how the duration calculus logic can be applied for the modelling of complex timed requirements in the domain, and adapt Quantified Discrete-time Duration Calculus (QDDC) [9] to perform a verification of WSTTS models under these requirements. The verification is carried out by first reducing the WSTTS models and the QDDC formulae to finite state automata [9], and then by encoding these automata in the language of the NuSMV [10] model checker. The latter is then invoked to verify the desired property.

The paper is structured as follows. In Sect. 2 we introduce the e-government case study that describes the problem of analysis of time-related properties. The formalism of WSTTS model, the representation of time-related properties, and DC logic are discussed in Sect. 3. Section 4 discusses the implementation of the analysis approach and experimental results, and Sect. 5 presents conclusions.

2 Case Study: e-Government Application

We illustrate our approach with the real e-government application. The goal of the application is to provide a service that manages user requests to open sites

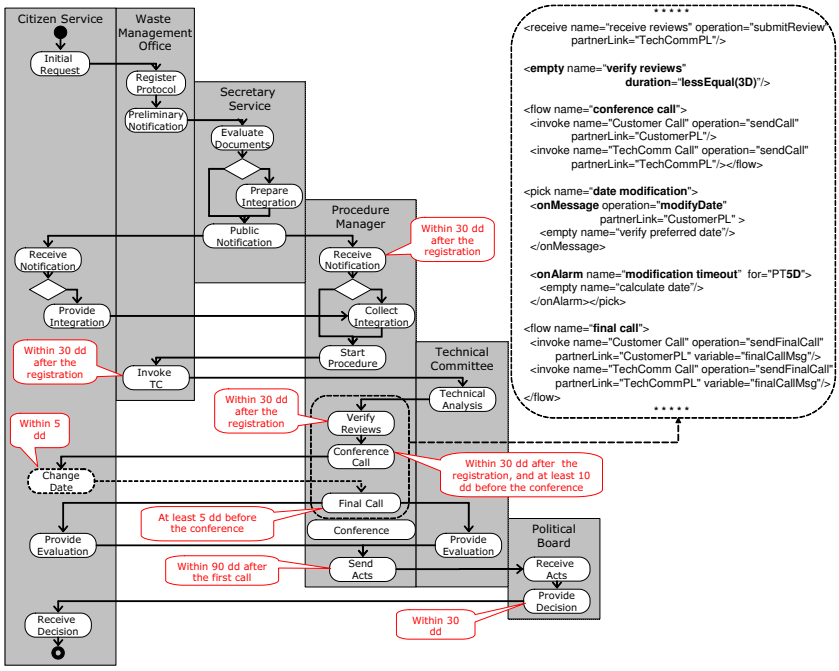

Fig. 1. Waste management application processes

for the disposal of dangerous waste. According to the existing Italian laws, such a request involves the interaction of different offices of the public administration, namely a Citizen Service, a Waste Management Office (WMO), a Secretary Service, a Procedure Manager, a Technical Committee, and a Political Board. In this application, the whole procedure is implemented as a composition of Web services that serve as interfaces to the processes of the above offices. We model the composition using BPEL specifications to describe partners interactions. The high-level choreography model of the request management procedure is presented in Fig. 1. The procedure describes different phases of the application management where the request is registered, the documentation is evaluated and collected, tha application is analyzed regarding the ecological impact of the site, the public conference is scheduled and organized, and final decision is provided.

Apart from the functional requirements, the execution of the process in the choreography should respect a set of timing requirements and constraints, dictated by Italian laws or by the agreement among the involved parties. These requirements (callouts in Fig. 1) specify, for example, that the period of time between the application registration and the notification of the `Procedure Manager` should not exceed 30 days, or that the participants can change the date within 5 days after the preliminary call.

The behavior of the composition and the possibility to satisfy these requirements depend on the time needed for the execution of the activities the involved parties are responsible for. The critical parameters here refer to the duration of

internal activities of the participants, and not to the communication time, which can be neglected.

The analysis of time-related aspects of the compositions requires explicit representation of timeouts, operation durations, and even complex properties expressing various timing requirements. While timeouts can be represented in BPEL, durations and timing requirements can not, and require specific way to be modelled. In our framework, we assume that the answer times are negligible by default, and that activities that have a non-negligible duration are annotated in the BPEL specification with an extra "duration" attribute. In Fig. 1 an excerpt of the annotated BPEL is represented. Here a BPEL event handler "date modification" is used to model 5-days bound for the conference data change. That is, the `onAlarm` activity is triggered if the user does not invoke the "modifyDate" operation within 5 days. On the contrary, the internal activity "verify reviews" is equipped with a duration annotation to express that certain time may be used for the reviews analysis. Timing requirements, however, can not be represented with durations associated to the activities and require more powerful notations. Consider, for instance, the requirement that the interval from the registration to the conference call should not exceed 30 days, and it is followed by the interval of length of at least 10 days, ending with the conference.

In order to be able to handle such aspects, it is necessary to provide model of the BPEL process behavior that allows for an explicit representation of time. Moreover, it is necessary to exploit and adapt the existing analysis techniques for reasoning about time to our problem domain. In the following sections we demonstrate how these issues can be addressed with the help of the WSTTS model, the duration annotations and duration calculus for complex time requirements.

3 Web Service Timed Transition System Model

In order to model the behavior of the BPEL process compositions, we propose the Web Service Timed Transition System (WSTTS) model, which adopts the formalism of *timed automata* for capturing the aspects specific to the Web service domain. In this formalism, the fact that the operation takes certain amount of time is represented by time increment in the state, followed by the immediate execution of the operation. In order to guarantee that the transition will take place at the right moment of time, the states and transitions of timed automata are annotated with the invariants and guards of the special clock variables.

Intuitively, WSTTS is a finite-state machine equipped with set of clock variables[3]. The values of these clock variables increase with the passing of time. A Web service composition thus is represented as a network of several such automata, where all clocks progress synchronously. In this model, the states of the WSTTS are equipped with the *state invariants* that express simple conditions over clocks and should be true when the system is in the state. Analogously, transitions are annotated with the set of *guards* that represent simple conditions

[3] It is also equipped with the set of non-timed variables of finite domains. For the sake of simplicity, we omit them in the formalism.

over clocks, and *resets* that are used to reset values of certain clocks to zero. The semantic of WSTTS is defined as a transition system, where either the time passes or a transition from one state to another immediately takes place.

Let X be a set of clocks. The constraints on the clock values $\Phi(X)$ are of the form $true \mid x \sim c \mid \phi_1 \wedge \phi_2$, where $\sim \in \{\leq, <, =, \neq, \geq, >\}$, $x \in X$, and $c \in \mathbb{T}$, a domain of time values.

Definition 1 (WSTTS). *WSTTS is a tuple (S, s_0, A, Tr, Inv), where*

- *S is the set of states and s_0 is the initial state;*
- *A is a set of input $?m$, output $!m$ or internal τ actions;*
- *$Tr \subseteq S \times A \times \Phi \times 2^X \times S$ is the set of transitions with an action, a guard, and a set of clocks to be reset;*
- *$Inv : S \to \Phi(X)$ assigns invariants to the states.*

In the definition, the effect of the transition from the state s to s' is to perform a communication or an internal action $a \in A$, and to reset set of timers to zero. The transition is possible only if the guard evaluates to true in the source state.

A clock valuation is a function $u : X \to \mathbb{T}$ from the set of clocks to the domain of time values. Let $\mathbb{T}_C$ denote a set of clock valuations. Let $u_0(x) = 0$ for all $x \in X$. We write $u \in Inv(s)$ to denote that u satisfies $Inv(s)$.

Definition 2 (Semantics of WSTTS). *Let (V, S, s_0, A, Tr, Inv) be a WSTTS. The semantics is defined as a labelled transition system $(\Gamma, \gamma_0, \to)$, where $\Gamma \subseteq S \times \mathbb{T}_C$ is a configuration, $\gamma_0 = (s_0, u_0)$ is an initial configuration, and $\to \subseteq \Gamma \times \{A \cup tick\} \times \Gamma$ is a transition relation such that:*

- *$(s, u) \xrightarrow{tick} (s, u + d)$, if $(u + d) \in Inv(s)$, and*
- *$(s, u) \xrightarrow{a} (s', u')$, if exists a transition $(s, a, \phi, Y, s') \in Tr$, such that $u \in \phi$, $u' = u[Y \mapsto 0]$, and $u' \in Inv(s')$.*

We define a Web service composition as a *WSTTS network*. The WSTTS network consists of n WSTTS P_i over common set of clocks X. The semantics of the network is given in terms of global timed transition system (GTTS). We use $\bar{s} = (s_1, \ldots, s_n)$ to denote a state vector, $\bar{s}_0$ to denote an initial state vector, and $\bar{s}[s_i/s'_i]$ to denote a state vector, where the element s_i is replaced by s'_i.

Definition 3 (GTTS). *Let $(X, P_1 \parallel \cdots \parallel P_n)$ be a WSTTS network.* Global timed transition system *has the form $(\Gamma, \gamma_0, \to)$, where $\Gamma \subseteq \langle S_1 \times \cdots \times S_n \rangle \times \mathbb{T}_C$, $\gamma_0 = (\langle s_{01}, \ldots, s_{0n} \rangle, u_0)$, and $\to \subseteq \Gamma \times \{A \cup tick\} \times \Gamma$ is a global transition relation:*

- *$(\bar{s}, u) \xrightarrow{tick} (\bar{s}, u + d)$, if $(u + d) \in \wedge_i Inv_i(s_i)$;*
- *$(\bar{s}, u) \xrightarrow{\tau} (\bar{s}[s_i/s'_i], u')$, if there exists $(s_i, \tau, g, Y, s'_i) \in Tr_i$, s.t. $u \in g$, $u' = u[Y \mapsto 0]$, and $u' \in \wedge_i Inv_i(s_i)$;*
- *$(\bar{s}, u) \xrightarrow{m} (\bar{s}[s_i/s'_i, s_j/s'_j], u')$, if there exists $(s_i, ?m, g_i, Y_i, s'_i) \in Tr_i$ and $(s_j, !m, g_j, Y_j, s'_j) \in Tr_j$, s.t. $u \in g_i \wedge g_j$, $u' = u[Y_i \cup Y_j \mapsto 0]$, and $u' \in \wedge_i Inv(s_i)$.*

In other words, the transition of GTTS is either a time passing transition, an internal transition of a particular WSTTS, or a shared communication action of two WSTTS.

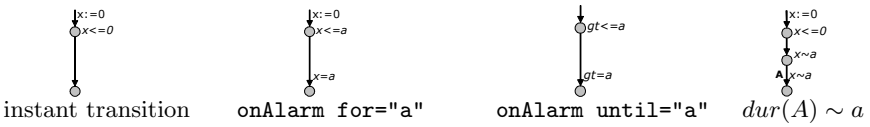

Fig. 2. Time-related constructs as WSTTS

Mapping BPEL constructs to WSTTS. We now give the definition of BPEL constructs in terms of the WSTTS formalism (see Fig. 2). We remark that, by default, all the activities of the underlying BPEL process are modelled as instantaneous. The fact that a particular activity may have certain duration is expressed explicitly through *duration annotations* that allow to specify bounds of the activity duration[4].

In this way all internal and message output activities are modelled as instant. Such a transition is semantically equivalent to adding an extra clock `x` to the source state of the transition, and the invariant of the state is `x<=0`. Hence, time can not pass in the source state of the instant transition. Input activities do not require such an addition, since they are blocked until corresponding output takes place, and therefore time can pass.

BPEL also defines activities that explicitly reference time. In particular, the activity `onAlarm` is used to represent timeouts and is modelled as an event handler. This activity has two forms: in the first it is fired when certain time passes (in Fig. 1 it is used in event handler to set up a 5 days timeout for the date modification); and in the second it is fired if the current absolute time has the specified value[5]. We model this absolute time using a special clock added to the WSTTS network model, namely *global_timer*. It is set to a certain value in the beginning of the execution, and is never reset later.

As we mentioned above, it is possible to specify certain duration for the activity. In this case the activity is explicitly annotated with the duration constraints (e.g. duration of activity "verify reviews" in Fig. 1). Such constraints are conjunctions of the clauses of the form $dur(A) \sim c$, where $\sim \in \{<, >, \leq, \geq, =\}$. In the WSTTS formalism this annotation is semantically equal to the sequence of two transitions. First transition is instant and it resets the clock `x`. The second transition and the intermediate state have the guards that evaluate to true, if the value of the clock `x` satisfies the duration constraints.

Specifying Time Requirements. While the constructs described above enable the explicit coding of simple time-related properties of WS compositions, we often encounter complex timing requirements which are hard to model with above constructs. Such requirements may express the time intervals between events (or a sequences of events), time bounds on some condition to hold or even complex logical combinations on them.

[4] We stress once more that our goal is to analyze the time properties that are critical for the business logic, and neglect the smaller "technical" times due, e.g. the communications.

[5] Another BPEL activity that deals with time is the activity `wait`. This activity is blocked for certain time period and is translated into WSTTS analogously

In order to express such properties we exploit a subset of duration calculus (DC) [11]. It allows us to express properties of finite sequences of behaviors and to measure the duration of a given behavioral fragment.

Let P range over propositional variables, D, D_1, D_2 over DC formulae, c over natural number constants, and $\sim \in \{<, \leq, =, >, \geq, \}$. The DC formula syntax is:

$$D := \lceil P \rceil^0 \ \mid \ \lceil\lceil P \rceil \ \mid \ D_1 \frown D_2 \ \mid \ D_1 \ \wedge \ D_2 \ \mid \ \neg D \ \mid \ len \ \sim \ c$$

DC formulas are evaluated over finite behaviors, i.e., over finite sequences of valuations of propositional variables $VAL(Pvar)^*$.

The constructs above have the following intuitive meaning:

- $\lceil P \rceil^0$ holds for the behavior consisting of a single state satisfying P;
- $\lceil\lceil P \rceil$ requires P to hold at all but the last states of the behavior;
- $D_1 \frown D_2$ splits the behavior into two subintervals, such that D_1 holds for the first subinterval, and D_2 holds for the second one;
- $D_1 \ \wedge \ D_2$ requires both formulas to hold, while $\neg D$ requires that D is not satisfied on the behavior;
- $len \sim c$ relates the duration of the interval with the constant value c.

Additionally, we write $\Diamond D = true \frown D \frown true$, if D holds for some subinterval of the behavior; $\Box D = \neg\Diamond\neg D$ denotes, that D holds for all subintervals.

The requirement that the interval from the protocol registration to the conference call should not exceed 30 days, and that from the call to the conference at least 10 days should pass, can be expressed with the following formula:

$$\Box(\lceil registration \rceil^0 \frown true \frown \lceil conference \rceil^0 \rightarrow (len \ \leq 30) \frown \lceil call \rceil^0 \frown (len \ \geq 10))$$

The formula says that every interval of the behavior that starts with the registration and ends with the conference consists of two subintervals with the call in between, such that the first lasts at most 30, and the second at least 10 days.

4 Implementation of Timed Analysis

We have implemented the ideas presented above as a prototype tool that allows for the timed analysis of Web service compositions. The tool inputs the initial composition of BPEL processes, enriched with the duration constructs, together with complex properties and translates it into the specification suitable for the formal techniques, such as model checking. In this implementation we adopt discrete model of time, and use (subset of) Quantified Discrete-time Duration Calculus [9] to express complex time requirements under this model. Under certain conditions the dense model of time may be analogously implemented.

The tool performs the transformation of the composition into the finite state automata representation and reflect the operational semantics of the global TTS given above. The clock variables are represented as global integer variable that synchronously increment their values when the time elapse transition happens. A special *tick* variable is used to denote this event. The results of [12] ensure that

for discrete time model the clock variables may be bounded without affecting the behavior of the system and therefore the resulting specification is finite.

The complex timing requirements are used in the composition analysis in the following ways. First, the composition specification M can be directly verified against a property represented with the QDDC formula D. For this purposes we construct an finite state automaton A that recognizes all and only the behaviors that satisfy the formula $\neg D$ ([9]). If the synchronous (i.e. lock-step) product $(M \times A(\neg D))$ of the specification and the automaton for the property negation is not empty, then the behavior of the composition violates the property D. Second, the complex time properties may express assumptions or constraints on the system behavior. In this case the tool builds a product $(M \times (A(D_1) \times \cdots \times A(D_n)))$ of the specification and the properties automata. This product restricts the specification model to behaviors that satisfies all the specified constraints. One can use this product for further analysis of the composition (e.g. for CTL or LTL model checking).

In order to illustrate the approach represented in the paper we have conducted a set of experiments on the analysis of the presented case study in different settings. In particular, in one set of experiments we assigned different bounds on the activity durations, and checked the composition for the deadlocks. Another set of experiments dealt with complex requirements expressing the bounds on intervals between various events (e.g. between application registration and conference call). The requirements were verified directly, or were used to model behavioral constraints. We have used the NuSMV model checker for verifying the corresponding finite state automata model. In the experiments the state space ranges from 10^{11} to 2×10^{12} states, and the verification time ranges from 0.6 to 15 seconds. Whenever the property is violated, the model checker generates a counterexample that represents an execution demonstrating the violation (e.g. a trace where both registration and conference call happened, but the time between these events was greater then the required bound).

5 Conclusions

We presented an approach for modelling and analyzing of time-related properties on Web service compositions defined by a set of BPEL processes. This approach is based on the formal model, Web Service Timed Transition System, that allows to take into account timed behavior of such compositions. We demonstrated how BPEL time-related constructs can be expressed in this formalism and presented a way to express time-related requirements using both simple modelling constructs or complex DC formulas particulary suitable for expressing such properties. The presented approach enables verification of WS compositions using model checking techniques.

The problem of the WS compositions analysis, in particular of BPEL processes, is investigated in the works of [4–7, 13]. While providing facilities for the verification of processes or their compositions, these approaches do not take time-related properties of composition behaviors into account. The work that is closer to ours

is presented in [14]. In this work, a formal model of BPEL processes, μ-BPEL, is presented that allows for mapping to a network of timed automata. However, [14] does not provide a way to explicitly specify the transition or state duration bounds, or complex time-related assumptions and requirements as those we model with DC formulas. In [15] temporal abstractions are exploited for the compatibility and replaceability analysis of Web service protocol. In that model one can specify when certain transitions must or may happen, similarly to what we achieve with our duration annotations. The work does not address the problem of the verification of these time properties and the abstractions are simple with respect to the set of properties we can express in our approach.

There are several directions for further research. In particular, we are working on the optimizations of the translations from BPEL to NuSMV code and applications of better analysis techniques that give a possibility to drastically improve the verification performance. Another line of research is to replace NuSMV with a model checker, such as UPPAAL, that can verify timed properties without requiring the generation of FSA.

References

1. Graham, S., Simenov, S., Boubez, T., Daniels, G., Davis, D., Nakamura, Y., Neyama, R.: Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. Sams (2001)
2. Khalaf, R., Mukhi, N., Weeravarana, S.: Service Oriented Composition in BPEL4WS. In: Proc. WWW'04. (2004)
3. Andrews, T., Curbera, F., Dolakia, H., Goland, J., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weeravarana, S.: Business Process Execution Language for Web Services (version 1.1) (2003)
4. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based verification of web service compositions. In: Proc. ASE'03. (2003)
5. Nakajima, S.: Model-checking verification for reliable web service. In: Proc. OOPSLA'02 Workshop on OOWS. (2002)
6. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In: Proc. WWW'02. (2002)
7. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: Proc. WWW'04. (2004)
8. Kazhamiakin, R., Pistore, M.: Parametric Communication Model for the Verification of BPEL4WS Compositions. In: Proc. WS-FM'05. (2005)
9. Pandya, P.: Specifying and Deciding Qauntified Discrete-time Duration Calculus formulae using DCVALID. In: Proc. RTTOOLS'01. (2001)
10. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model checker. Int. Journal on STTT (2000)
11. Chaochen, Z., C.Hoare, Ravn, A.: A Calculus of Durations. In: IPL. (1991)
12. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. (1994)
13. Pistore, M., Roveri, M., Busetta, P.: Requirements-Driven Verification of Web Services. In: Proc. WS-FM'04, ENTCS. (2004)
14. Geguang, P., Xiangpeng, Z., Shuling, W., Zongyan, Q.: Towards the Semantics and Verification of BPEL4WS. In: Proc. WS-FM'04, ENTCS. (2004)
15. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On Temporal Abstractions of Web Service Protocols. In: Procs of CAiSE Forum. (2005)

Invocation Order Matters: Functional Feature Interactions of Web Services

Michael Weiss[1], Alexander Oreshkin[1], and Babak Esfandiari[2]

[1] School of Computer Science, Carleton University, Canada
weiss@scs.carleton.ca, oreshkin@comnet.ca

[2] Department of System and Computer Engineering, Carleton University, Canada
babak@sce.carleton.ca

Abstract. This paper proposes a method for detecting feature interactions related to the functionality of a composite web service. There are several important ways in which functional features of Web Services can affect each other through interaction. A feature interaction is an undesirable side effect of the composition of services (also known as features in this context). There are various causes for interactions, including race conditions, violation of assumptions, goal conflicts, and invocation order. We have categorized the sources of feature interactions among web services in related work. In this work, we present the results of ongoing work on the formalization of functional interactions between web services. In our approach we use on labeled transition systems to model service compositions. These models are analyzed using the LTS Analyzer to detect undesirable feature interactions. As a specific example, we look at invocation order as a source of functional feature interaction.

Keywords. Validation and verification, service composition, causes of feature interaction, invocation order, and labeled transition systems.

1 Introduction

Service-oriented approaches promise to provide businesses with the freedom they need to serve their customers no matter what software or hardware configuration their clients are using. With this technology, businesses would be able to adapt quickly and easily to changes that occur both on the client as well as the business-side.

Business services are implemented as functional software features. These features are then made accessible to other businesses as distributed software components. (Of course, services also have non-functional properties, but it is not yet standard practice to include these in the service interface.) However, rapid changes in the services due to the dynamic nature of businesses can lead to undesirable results and poor service quality, when these services are interacting with each other in undesirable ways.

In the literature this problem has been studied as the feature interaction problem. The problem of undesirable interactions between components of a system can occur in any software system that is subject to changes. However, the problem itself and

approaches to address it have been largely unknown outside a small community of people specialized on the design of telecommunication switches. However, some progress has been made recently towards explicitly modeling and analyzing feature interaction in other domains [4, 5]. Undesirable side effects of web service composition as feature interactions have first been described in [6], and further developed in [7]. In [8], we presented a classification of feature interactions among web services.

Feature interactions are interactions between independently developed features, which can be either intended, or unintended and result in undesirable side effects. In [6] we make a distinction between functional and non-functional interactions. This distinction reflects that many of the side effects affect service properties such security, privacy, or availability. However, our focus in this paper is on functional feature interactions, which have not been covered in earlier work on feature interactions among web services. These include race conditions, violation of assumptions, goal conflicts, and invocation order. These are also the categories of interactions that have traditionally been studied by the feature interaction community.

The hierarchical architecture of building larger services from smaller services, together with object-oriented principles such as encapsulation and information hiding, creates many challenges in dealing with service interactions. It is thus desirable to develop formal approaches to modeling web services and detecting problematic interactions. The approach presented in this paper is based on Labeled Transitions Systems (LTS). Similar work has been presented, for example, in [1]. What our work adds is a model of the type of interactions to expect, and ways of detecting them.

The interaction models are analyzed using the LTS Analyzer (LTSA) [3] for violation of properties that we can specify. The LTSA tool uses well-established model checking techniques based on state-space exploration to automatically analyze properties of models. This approach lays a foundation for developing a formalized methodology to address the feature interaction problem in web services.

2 Feature Interaction Problem

The feature interaction problem first appeared in the domain of telecommunications [2]. The problem concerns coordination of feature interactions such that their cooperation yields a desired result. Many hundreds of features can interact directly or indirectly and can affect each other's behavior. Some interactions are desirable, while other interactions can lead to undesirable side effects such as an inconsistent system state, an unstable system, or data inaccuracies.

As web services technology matures, it is becoming crucial to manage the interactions among web services. The feature interaction problem is presenting new challenges for the web services domain. Causes for functional feature interactions in web services have been categorized as following [8]:

- **Race conditions.** A race condition occurs between multiple components of a composite web service (which we will refer to as feature in line with the feature interaction terminology), when the outcome of executing the service depends on timing delays (also known as glitches) between the executing of features.

- **Violation of assumptions.** Web service developers need to make some assumptions about how a web service will be used by service consumers (including other web services). When service consumers break those assumptions, the service may no longer operate correctly. Similarly, the expectations of consumers may be violated by the implementation of a service.
- **Order of invocation.** The correct operation of a composite web service may also depend on the order of invocation of some of its features. The service may assume a certain order in which events will take place. If a service consumer breaks this order, the correctness of the results is no longer guaranteed.
- **Competition for resources.** Service consumers may be competing with each other through access to limited resources on a service provider. Examples of such resources are: disk space, memory, CPU, network bandwidth, database access, etc. The correctness of one consumer may be compromised by the interference of tan other that is using more than its share of resources.
- **Goal/policy conflicts.** Each feature has a specific task or goal it is trying to achieve, or policy that is follows. When there is only one web service, there is one goal or policy. However, when services are combined into a higher-level service, each with its own goals or policies, it may be that the goals or policies of these services are in conflict, and we cannot guarantee their achievement.
- **Encapsulation/information hiding.** If encapsulation is used, service consumers are not aware the inner workings of service providers. This necessarily means that consumers must make some assumptions about providers. If those assumptions are wrong, the correct operation of the service is questionable.

3 Feature Interactions in a News Service

To provide a concrete idea of how functional feature interactions can arise, we will look at a specific interaction of web services. For the example, we first present the web services involved, giving a description of each web service and its main features. Then, we illustrate and discuss the feature interaction problems arising.

This is one of several scenarios we looked. The other scenarios included a reservations system, a remote environment management system, and a supply chain. Each example illustrated different types of interactions: race conditions, violation of assumptions, conflicting goals, information hiding, and order of invocation. However, in this paper, we deliberately focus on order of invocation as one type of interaction for a better exposition of the approach in the limited space provided.

Consider a News service that provides clients with access to full-text articles on a pay-per-view basis. It uses a News Catalog service as a source of recent headlines and articles. A service that wants to use the News Catalog service must also use a Logging service associated with News Catalog. It needs to provide payment information with each request for the body of an article, which will be logged. At the end of each billing period, News Catalog will use the log maintained by the Logging service to compile a statement and charge the client's credit card (or similar) for their usage.

News Catalog provides two features, Get Headlines and Retrieve Article. The features of the Logging service are Log and Get Log. The News service provides the Get News

and Get Article features. It also has a Cache feature, which is used in the implementation of Get Article. The intent of this feature is to avoid charging a user more than once for retrieving the same article, and to speed up the retrieval of full text articles. It maintains a single cache of the most recently retrieved articles. When a client requests an article, the Get Article feature executes. If the article is not in the cache, an external request is made to the News Catalog service, and the article is retrieved and subsequently cached. If the article is in the cache, then it is immediately returned to the client, and no external request to the News Catalog service is made.

As described, News Catalog relies on the information provided in the log maintained by Logging. As part of the usage contract between News Catalog and a service consumer such as News, the service consumer must record all article requests made by the client with Logging. News Catalog relies on this log for charging clients for their usage. However, the News service also contains a Cache feature, which saves recently accessed articles. When a user requests to read an article that happens to be saved in the cache, the article is retrieved from the cache instead of from the News Catalog.

A manual analysis of this scenario tells us the only those client requests are being logged for which the requested article is not found in the cache. In the article is found in the cache, the requests are not being logged. What is worse is that no one is charged for those requests except the very first client that caused this article to be cached. So it is possible to gain access to an article for free (as long it stays in the cache). This is an example of a feature interaction due to an incorrect *invocation order*. The behavior of the service depends subtly on when requests are logged. This could occur either before (no interaction), or after the cache is inspected (interaction).

The situation demonstrated here also demonstrates another type of undesirable feature interaction between services, which occurs due to the *information-hiding* nature of web services. The essence of the problem from the point of view of the News Catalog service is that it is unaware of the Cache feature present in one of its service consumers, the News service. It (incorrectly, it turns out) expects that all requests for articles made by the clients of the consumer services will be logged with the Logging service associated with the News Catalog. It does not expect that these consumer services have mechanisms that will prevent some of the requests from being logged.

4 Formal Analysis of Feature Interactions

For larger composite web services, which may exhibit many potential feature interactions, the manual analysis described for the example is not feasible. It is, therefore, desirable to perform the detecting problematic interactions using formal approaches. Our approach is based on Labeled Transitions Systems (LTS), a form of state machine for the modeling of concurrent systems, in which transitions are labeled with action names. For small systems a LTS can be analyzed using a graphical representation of the state machine description, but for large number of states and transitions, an algebraic notation for describing process models is required.

Such a notation is provided by FSP (Finite State Processes) [3]. The LTS Analyzer (LSTA) [3] supports the analysis of a system described in FSP to verify that the LTS model satisfies specified safety and progress properties. Informally, a property is an

attribute of a program that is true for every possible execution of that program. A *safety property* is a statement of what is considered to be a correct execution of the system. If anything happens in the system that goes against the specifications of the safety property, the system is considered to be in error. A *progress property* asserts that some part of the system will eventually execute. A common example of a violation of this property is a deadlock. The analysis of a system is based on (exhaustive) state-space exploration. Its main benefit is that can be automated, thus avoiding the inherent error introduced when using manual methods.

An FSP model comprises a collection of constant definitions, named processes, and named process compositions. FSP offers rich syntactic features including guards, choices, variables, and index ranges. It supports action non-determinism and process parameters. In the LTS analysis that follows, our goal is to detect the invocation order problem given a model of the News service in FSP, and a definition of safety properties that have to hold. Fig. 1 shows an FSP model of the News service features.

```
// Logging service
LOG = (log_request-> LOG | get_log-> LOG).

// News Catalog service
CATALOG = (get_article-> CATALOG | proccess_billing-> BILLING),
BILLING = (log.get_log->process->CATALOG).

// Cache feature for the News service
CACHE = (add_article->NON_EMPTY_CACHE),
NON_EMPTY_CACHE = (add_article->NON_EMPTY_CACHE |
 retrieve_article->NON_EMPTY_CACHE).

// The News service
NEWS_SERVICE = (request_article-> CHECK_CACHE),
CHECK_CACHE = (cache.found->cache.retrieve_article->NEWS_SERVICE
 | cache.not_found->ACCESS_CATALOG),
ACCESS_CATALOG = (log.log_request->catalog.get_article->
 cache.add_article->NEWS_SERVICE).
```

Fig. 1. FSP model of the News service features

The main concern from the point of view of correct logging is, that for every article request there should be an invocation of the Logging service, which will log that article request. We express this concern as a safety property as shown in Fig. 2.

```
// Check that each request for an article is logged.
property P = (request_article->log.log_request->P).
```

Fig. 2. Safety property for the News service

Finally, the composite process that represents the News Catalog service and the News service interacting is expressed simply as shown in Fig. 3

```
// Composite process that represents the News service
||NEWS = (NEWS_SERVICE || cache:CACHE || catalog:CATALOG ||
 log:LOG || P).
```

Fig. 3. Result of composing the News Catalog and the News service features

The composite process includes the safety property `P`. Using the LTSA, we can perform a safety check analysis to see whether the property can be violated. The trace will tell us, if there is a sequence of events, where a client request for a news article is not properly logged. Performing this safety check, we get the result in Fig. 4.

```
Trace to property violation in P:
     request_article
     cache.not_found
     log.log_request
     catalog.get_article
     cache.add_article
     request_article
     cache.found
     cache.retrieve_article
     request_article
```

Fig. 4. Result of safety check analysis

The trace of events demonstrates that there is a situation, where the safety property is violated. This happens when a client requests an article, and then another client requests the same article. Since the article was cached after the first request, the next time it was retrieved from the Cache instead of from the News Catalog service. This analysis reveals that the cache feature of the News service can cause problems, when it is paired with the incorrect order of the Logging service invocation.

The graphical representation of the News service can provide us with additional insight in the scenario that led up to the feature interaction. Fig. 5 shows the LTS for the News service, indicating (in red) the violating transition (`request_article`).

Although our focus in this paper was on order of invocation types of interactions, the situation demonstrated at hand can also be characterized as an undesirable interaction that occurs due to the encapsulation or information hiding nature of web services. The essence of the problem from the point of view of the News Catalogue service is that it is unaware of the Cache feature present in one of its service consumers, the News service. It expects that all article requests made by the clients of the consumer services will be logged with the Logging service associated with the News Catalogue. It does not expect that these consumer services have mechanisms that will prevent some of the requests from being logged. This is an example where an interaction can be described as both a functional, as well as a non-functional feature interaction.

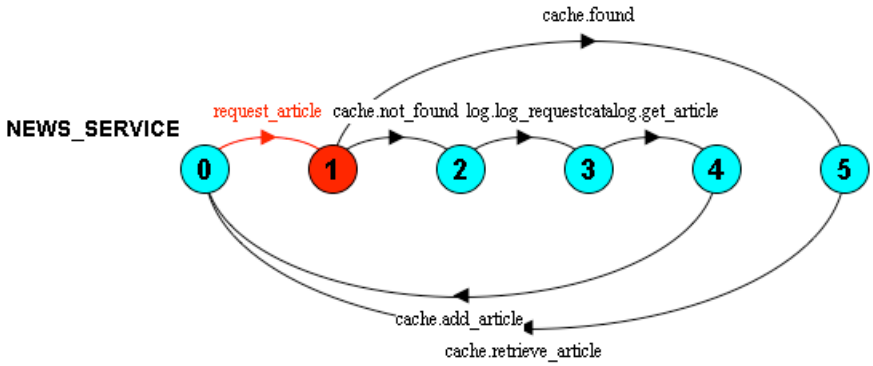

Fig. 5. LTS for the News service

5 Conclusion

LTS modeling of web services can be a significant aid in developing a formal methodology for functional feature interaction problems detection. Rather than modeling the feature interactions explicitly, it was only necessary to model the main aspects of behaviour of each web service, and to define safety properties that detect assumption violations, race conditions, and incorrect order of invocation types of interactions.

One advantage of using this approach is that we can detect potential feature interactions before web service implementation. Any undesirable interaction that is identified at this stage saves us the cost of correcting those errors during the implementation. By developing LTS models of web services and instrumenting them with appropriate safety and progress properties we are able to detect interactions automatically using the LTSA tool. LTSA can not only detect the presence of a feature interaction problem, but also find the exact sequence of events that leads to the problem.

A well-known problem with LTS analysis is the explosive nature in which the number of states increases. This may limit the size of systems that can be analyzed, even after optimizations such as state hiding [3] have been applied. Another issue is that the safety and progress properties have been developed to target specific known interaction problems. However, currently there is no systematic process for developing these properties to detect unknown interaction problems. Here, we rely on the completeness of our understanding of the types of interactions that can occur. Also, algebraic notations such us the pi-calculus should be explored.

Future work will include the application of the approach to larger case studies. For example, we are currently investigating its application to a supply chain. The potential interactions in these kinds of systems are significantly more complex, in particular, because we are not focusing on isolated interactions, one at a time. In our earlier work, results were also determined for the other classes of feature interactions problems (that is, race conditions, violation of assumptions, competition for resources, goal conflicts and information hiding), but mostly in isolation. We are also interested

in extending the work on offline detection to the detection of feature interactions at runtime, and ultimately, their runtime resolution. This would be of great relevance to situations where web services are dynamically composed.

References

1. Foster H., Uchitel S., et al, Compatibility Verification for Web Service Choreography, International Conference on Web Services (ICWS), IEEE, 738-741, 2004.
2. Keck, D, and Kuehn, P., The Feature and Service Interaction Problem in Telecommunications Systems, IEEE Transactions on Software Engineering, 779–796, 1998.
3. Magee, J., and Kramer, J., Concurrency: State Models and Java Programs, Wiley, 1999.
4. Pulvermüller, E., Speck, A., et al (2001), Feature Interaction in Composed Systems, ECOOP Workshop on Feature Interactions in Composed Systems, Technical Report 2001-14, 1-6, Universität Karlsruhe, Fakultät für Informatik.
5. Turner, C.R., Fuggetta, A., et al, A Conceptual Basis for Feature Engineering, Journal of Systems and Software, 49:1, 3-15, December 1999.
6. Weiss, M., and Esfandiari, B., On Feature Interactions among Web Services, International Conference on Web Services (ICWS), 88-95, IEEE, 2004.
7. Weiss, M., and Esfandiari, B., On Feature Interactions Among Web Services, International Journal of Web Services Research, 2(4), 21-45, October-December, 2005.
8. Weiss, M. and Esfandiari, B., Towards a Classification of Web Service Feature Interactions, International Conference on Service-Oriented Computing (ICSOC), LNCS, Springer, 2005 (to appear).

Behavioural Verification of Service Composition

Pascal André, Gilles Ardourel, Christian Attiogbé

LINA - CNRS FRE 2729, University of Nantes
Gilles.Ardourel@univ-nantes.fr

Abstract. In this work, we present a model for the definition, the composition and the verification of services. In this model, the services are bound to components. The composition of components results in the interaction of services. An inconsistency in this interaction foretells the failure of the service composition. The contributions of this work are: *i)* a formalism that allows a flexibility in the description and the composition of services: optional behaviours, optional sub-services and renaming; *ii)* service interfaces enriched with behavioural informations that contribute to the early detection of service incompatibilities; *iii)* the verification of the behavioural compatibility of service composition. The verification process is based on LTS techniques; we reuse existing verification tools.

1 Introduction

The development of large scale applications requires modular approaches such as Service Oriented Computing [11, 12] or Component Based Software Engineering [9]. In both approaches, the success depends on the availability of: expressive languages for the elements (services and components) and their composition [2, 11], tools for checking the correct usage of the elements, and tools for managing reliable element libraries. As a partial answer, we propose a component model to describe services and service composition. One component offers services which may be called by another component service. We also study the means for verifying the correct usage of these services. Indeed, it is important to detect the defects which could lead to a faulty behaviour of a developed system early in the development. A bad interaction between a called service and the calling one may lead to a blocking of the whole system. To ensure the desired properties (correctness, compatibility, composability...) we need formal descriptions of the services. In our model, the service providers are explicitly represented as components but service implementation is left out. The use of an abstract formal model makes it possible to hide the implementation details of the components in order to have general reasoning techniques which are adaptable to various implementation environments. The EJB and CORBA-based approaches [6] are not dealt with.

Our approach is based on a simple formalism for modelling and composing services and components. A component is viewed and used only through its services which constitute its behavioural interface. The use of service is central to

the verification of compatibility when assembling components because the components are "connected" by their services. The contributions of this work are: a formalism that allows a flexibility in the description and the composition of component and services: optional behaviours, optional sub-services and renaming; service interfaces enriched with behavioural informations that contribute to the early detection of service incompatibilities; the verification of behavioural compatibility of service composition.

This article is organised as follows. Section 2 overviews the kmelia formalism. Section 3 describes the composition of services in our formalism and the desired properties. Section 4 details the verifications that are done during the composition. We conclude by a short discussion in Section 5.

2 The Kmelia Model

The Kmelia model is a component model based on services. A Kmelia specification describes services, components and compositions. A service encodes a functionality. A component provides services and may require other services (to supply the provided services). A composition links a set of components by their services.

The main features of Kmelia are: the encapsulation of services in components; the enrichment of interfaces that allows an early detection of service composition errors; the flexible description of the service behaviour.

2.1 Component, Services and Sub-services

In Kmelia the service is the basic concept (services perform functions) while the component is a modular structuring unit that encapsulates services and allows a fine control of when and by which services they can be called. The separation between services and components allows system models with partially supported services (some services work and others do not). The component defines a shared entity for services (namespace, data, sub-services, constraints). The component provides an *interface* made of *provided services* and *required services* (from some abstract service provider). Consequently the component defines internal services.

Kmelia defines some minor services (called *sub-services*) that can only (and optionally) be called during a transaction with another service. These sub-services can be shared in a component. Sub-services can be used in place of parameters or to introduce some flexibility in a service protocol. For instance in Figure 1, the `main` service requires a `chat` service that can ask for a password. The use of a `passw` sub-service has several benefits:

- it separates the password protocol from the main service protocol, enhancing readability and reuse;
- it allows the control of the points where the password can be asked;
- it can be easily replaced by another password protocol;
- it allows the service to work even if the password is not asked.

2.2 Enriching Interfaces of Services

The interface of a service `serv` is made of a signature, a precondition, a post-condition and sets of service names (*subprovides*, *extrequires*, *calrequires*...). The *subprovides* set contains the names of the sub-services.The two others are two kinds of required services names. In Kmelia we distinguish two kinds of service dependencies: external and caller dependencies. The former (*extrequires*) is quite usual and establishes that a service needs other ones in order to work. The former (*calrequires*) is explicitly required from the caller of the service `serv`.

Following is the code of the interfaces of the service `main` which requires a `chat` service and provides a sub-service `passw`, which in turn needs an identification service (`idserver`) from its caller.

```
Service main
Interface
   subprovides : {passw}
   extrequires : {chat}
...
end
```

```
Service passw : String()
Interface
   calrequire : {idserver}
...
end
```

2.3 Specifying Flexible Behaviours with eLTS

A service is formally specified with a 5-uple $\langle \sigma, P, Q, V, \mathcal{B} \rangle$ where σ is the service signature, P is the precondition, Q is the postcondition, V is the set of local variables and $\mathcal{B}$ is the extended labelled transition system (**eLTS**) which describes the service behaviour.

The service behaviour $\mathcal{B}$ provides details on the interactions between services and the order in which these interactions may occur. Formally $\mathcal{B} = \langle S, \mathcal{L}, \delta \rangle$ where S is the set of states, $\mathcal{L}$ is the set of transition labels and $\delta \in S \times \mathcal{L} \rightarrow S$ is the transition relation. The labels on transitions (which concrete form is: `is--lab-->fs`) are combinations of actions which may be internal actions or interactions. An internal action is a computation (*elementary action* or a composition of internal actions) that does not involve other services. An interaction denotes an exchange on the service link (service channel). In Figure 1, the composition defines such a link between the required service `Client.chat` and the provided service `Server.chat`.

The syntax of an interaction is: `channel(??|!!|!|?)message(param*)`; it is inspired by the Hoare's CSP language. The message can be a service call (??), a service result (!!) or a synchronous communication (send !, receive ?). When writing a behaviour, one does not know which components will communicate, but one has to know the channel on which it will take place. The channel is defined when the components are composed but its name depends on the service interface. The placeholder keyword `CALLER` is a special channel that stands for the channel opened for a service call, otherwise it is the required service name. The following descriptions are the behaviours of the services `main` and `passw` in the Kmelia syntax.

```
Service main
Variables  # local to the service
  c:Boolean
Behavior
init e0    # e0 is the initial state
final e6   # e6 is a final state
{ e0 -- _chat!!chat --> e1,
      # invocation of the chat service on its channel (named chat)
  e1 -- _chat!login(myLogin) --> e2,    # sending the login
  e2 <passw>, # specifies callable sub-services on node e2
  e2 -- c:=_chat?cnx --> e3,
      #ask for the result of the connection
  e3 -- [not c] nop --> e5,
  e3 -- [c] _chat!message("hello world") --> e4,
  e4 --  _chat!message("\stop") --> e5,
  e5 -- _chat??chat --> e6 #wait for end of service chat
}
end

Service passw () : String
# gives the client's password to a trusted server
Variables # local to the service
  trustserv : boolean,       #is the server a trusted one ?
  id : integer
Behavior
init e0    # e0 is the initial state
final f    # f is a final state
{ e0 -- __CALLER!!idserver --> e2,
  e2 -- __CALLER??idserver(id) --> e3,
      # calls the identification service of the caller
  e3 -- trustserv:=isTrusted(id) --> e4,
      # isTrusted is an elementary action here
  e4 -- [trustserv]__CALLER!!passw(myPwd) --> f,
  e4 -- [not trustserv]__CALLER!!passw("") --> f
      # sends the code as a result of the service
}
end
```

The `_chat!!chat` is a (required) service call on the implicit `chat` channel. The `__CALLER!!passw(myPwd)` is a result of service. You can see a notable difference between the above code and the eLTS of Figure 1: all channels are explicit. In many cases, channels can be omitted and deduced either from the context or from default rules. This syntactic sugar is not currently implemented in our prototype.

In an eLTS the states may be annotated with sub-services. It means the sub-services may be launched from this state and the control returns to this state when the launched sub-service is terminated; for example calls to the service `passw` are enabled only in the third state of the `start` service. Such a notation allows flexibility (optional sub-services), service sharing and LTS size reduction.

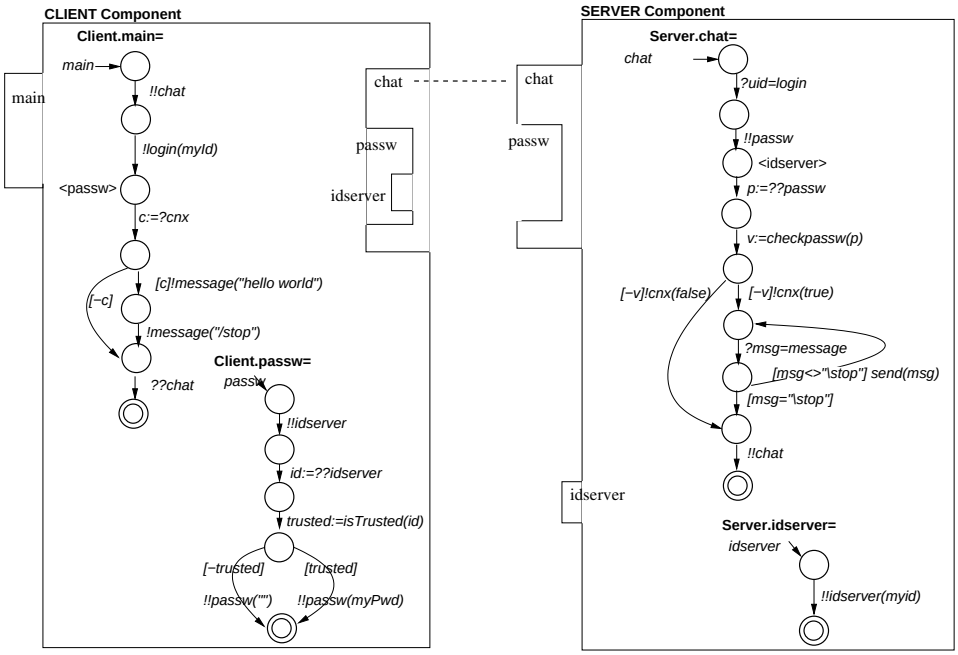

Fig. 1. Composition of chat services of two components

3 Service Composition

In Kmelia we distinguish two kinds of composition. The first one encapsulates services in a single unit while the second one composes services from different components.

Encapsulation of Services in a Component A component that encapsulates several services should have the following properties: at least one service should be provided and it should be visible; every service of the unit should be callable, either directly or indirectly (via another service); every sub-service in a service interface must be callable at some point during the service evolution; no service can have itself as a sub-service; the sets of external required services and caller required services of a service must be disjoint.

Composition of Components Two or more components may be composed by linking their provided and required services to build larger components. Therefore a composition of services is the result of the link of the required services and provided services from different components. The composition of services is the support for interaction between the components. It is only at composition time that the channels are resolved according to the links declared between the components. Each linked service is then defined as the communication context of an other.

For the composition to be effective, several properties should be satisfied: *(Static) Interoperability properties*: compatibility of signatures and interfaces (naming, typing and compatibility of pre and post conditions); *Behavioural compatibility*: absence of deadlocks in the communication.

4 Behavioural Verification of Services

Formal verification of services may be performed according to various aspects. We generalise the three levels of interoperability of Yellin&Strom [13] to four levels of conformity: service signatures, enhanced service signatures (sub-services may be participant of a service), contracts (pre/post conditions) and behaviours (the correct interactions between the caller service and the called service). In the following, we focus the verification on static interoperability (level 1 and 2) and behavioural compatibility (level 4).

4.1 Interface Compatibility

The first step of the verification of the behavioural compatibility is a pairwise comparison of behavioural interfaces. In Kmelia the interface of a component contains the sets of provided and required services (with the naming and typing informations); additionally, informations on required or called sub-services are attached to the service interfaces. In a similar way, these informations are available for the service descriptions. Accordingly, the static analysis of the interface of a component is achieved by using: *i)* simple correspondence checking algorithms and possibly standard typing algorithms; *ii)* deep investigation on the availability of required or called sub-services. At this stage, we can detect several incompatibilities such as missing sub-services, signature mismatch...

4.2 Behavioural Compatibility

The *behavioural compatibility* between services is a widely studied topic [13, 7, 3]. Behavioural introspection (discovering the component behaviour) is one way to deal with behavioural compatibility; but one has to prove compatibility. Checking behavioural compatibility often relies on checking the behaviour of a (component-based) system through the construction of a finite state automaton. However the state explosion limitation is a flaw of this approach [3].

In Kmelia the behaviour of the component relies on the behaviours of its services. Therefore the component interacts correctly with its environment if its services are *compatible* with the other services. The main concern is to check that a given service interacts correctly with another one (which may be provided by a third party developer). The interaction between services may involve not only two but many services. But we consider only one caller service and one called service at time. Remind that each service is described with a transition system; the transitions are labelled with: service calls, elementary actions, guarded actions and communication actions (messages).

A service is *behaviourally compatible* with another one if their eLTSs are matching: they evolve independently or they perform complementary communication actions. That is the basis of our compatibility analysis approach. This approach is the widely used one [13, 3, 4]; but we adapt it to a more expressive LTS used in our model. A complete interaction between the services of several components results in a pairwise local analysis between the eLTS of a caller and that of the called service. Indeed, two eLTS interact until a terminal state if their labels are in correspondence according to a protocol that we have defined. The protocol is a set of rules based on the labels of the transitions available from a current state. The rules indicate the correct evolutions according to the current states of involved services and the labels of the transition: either independent evolution or a communication involving complementary actions from the peers. After a final state of a called service, the caller may continue with independent transitions or with transitions that imply other (sub-)services.

The ideas presented here are put into practice with the LOTOS/CADP Toolbox [8]. We defined a policy to translate our service behaviours into LOTOS processes. In this translation, all interactions are considered as communications between processes. Thereafter we use the LOTOS communication facilities to deal with behavioural compatibility. A similar experiment has been conducted with the MEC[1] tool.

5 Discussion and Perspectives

We have presented an abstract component model and a formalism that permit the flexible description of interacting services which are defined as extended labelled transition systems. This model supports service composition and component composition. A *chat* example involving two components is presented and it illustrates an interaction between composed services and their accompanying sub-services.

Unlike most of existing approaches [13, 3, 5] where a component has a behaviour (called a protocol), we argue for a model were the provided services, defined as LTS, have their own behaviour. This moves up the granularity for the use of components and increases the usability of components by considering a service level. When our service behaviours are reduced to combinations of messages, we get the low level of usability found in the aforementioned approaches.

The study of compatibility at the component behaviour level is central to CBSE approaches and has motivated number of works [13, 7, 3, 5] and applications to web-services [4]. We build on these works but we extend the study to encompass the granularity considered here for services and components. Our approach allows for a local verification of the behavioural compatibility between composed services. Experiments are performed with the approach using existing toolboxes. Compared to related works [3, 10], our approach works at the abstract specification level, it offers a more flexible formalism than the ones proposed by [13, 3, 4] for the description of interacting services. We adopt a pairwise verification approach that avoids state explosion like in [3]. We can extract several

collaborations a la Yellin&Strom [13] from a single of our service behaviours which interweaves collaborations on different channels and allows optional calls of services.

The perspectives of this work are: to reinforce the correctness properties of component with supplementary study of correctness of components and services with regard to their environment; to extend the COSTO (Component Study Toolbox) prototype under development to cover mechanised analysis concerns. The prototype already integrates parsers, translators to LOTOS and MEC, static and dynamic interoperability checkers. However we lack a graphical interface to guide and assist the user. Then we will propose an open source delivery of the toolbox.

References

1. P. Crubillé A. Arnold and D. Bégay. *Construction and Analysis of Transition Systems with MEC*. AMAST Series in Computing: Vol. 3. World Scientific, 1994. ISBN 981-02-1922-9.
2. M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors. *Service-Oriented Computing*. ACM, 2004.
3. P. Attie and D. H. Lorenz. Correctness of Model-based Component Composition without State Explosion. In *ECOOP 2003 Workshop on Correctness of Model-based Software Composition*, 2003.
4. L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In *TES*, pages 15–28, 2004.
5. A. Bracciali, A. Brogi, and C. Canal. A Formal Approach to Component Adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.
6. C. Canal, L. Fuentes, E. Pimentel, J. M. Troya, and A. Vallecillo. Adding roles to corba objects. *IEEE Trans. Softw. Eng.*, 29(3):242–260, 2003.
7. L. de Alfaro and T. A. Henzinger. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE)*, pages 109–120. ACM Press, 2001.
8. J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In R. Alur and T. A. Henzinger, editors, *Proc. of the 8th Conference on Computer-Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 437–440. Springer Verlag, 1996.
9. G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. A. Szyperski, and K. C. Wallnau, editors. *Component-Based Software Engineering, 8th International Symposium, CBSE 2005, USA, May, 2005*, volume 3489 of *LNCS*. Springer, 2005.
10. P. Inverardi, A. L. Wolf, and D. Yankelevich. Static Checking of System Behaviors using Derived Component Assumptions. *ACM Transactions on Software Engineering and Methodology*, 9(3):239–272, 2000.
11. M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISE*, pages 3–12. IEEE Computer Society, 2003.
12. M. P. Papazoglou and D. Georgakopoulos. Introduction to service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.
13. D.M. Yellin and R.E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.

Static Validation of Business Process Compatibility in Web Services Choreographies

Agustín Cernuda del Río, Jose Emilio Labra Gayo, Daniel Gayo Avello, and Daniel Fernández Lanvín

Departamento de Informática*, Universidad de Oviedo - Asturias (Spain)
{guti,labra,dani,dflanvin}@uniovi.es

Abstract. Many technologies related to software components have a mostly descriptive purpose. It seems desirable to promote automatic specification and validation strategies, developing techniques that, from the descriptions, are able to detect defects in a static manner (before execution time). Technology transfer is also a main issue to encourage the adoption of such technologies. It is also necessary to incorporate such validations into the design and development process, towards the ideal of correct-by-construction software.
Recent work on web services choreographies may be combined with simple and affordable verification techniques, making possible to exploit the descriptive information of the choreographies for the described analysis, by means of systems which are relatively easy to build.

1 Introduction

Software components initiatives such as COM, CORBA, JavaBeans or web services solved basic interoperability infrastructure problems, but did not go too far in the static verification of component compatibility [1]. Only signature compatibility is verified in those models; this was done already in precedents such as OSF-DCE, by means of the IDL language. In previous work we have developed a model, Itacio [2] for the static verification of software components, which allows any facet of a component which is suitable for being expressed by means of Constraint Logic Programming (CLP) to be denoted and used in an automatic verification system.

At a much higher, coarser-grained level, Service-Oriented Architectures (SOA) are also a scenario for compatibility verifications. The development of standards for describing these interactions shows a familiar evolution pattern: there is a high degree of consensus and adoption of the basic interoperation standards (HTTP, XML, SOAP, SWDL...) but the scene is less clear in the upper layers, where several proposals have been made (XLANG, BPML, WSCL, WSCI). They seem to be reaching maturity in two main lines: BPEL4WS ([3], [4], [5] and

* This work has been funded by Seresco, S.A. and the Principality of Asturias (project reference: FC-04-PC-39)

WS-CDL [6]. Reasonably, the development of these standards is focusing on notation and expressive power, and not on static verification. This paper proposes a way for verifying protocol compatibility from WS-CDL and/or BPEL4WS specifications, and for guiding the development process with such information.

2 Web Services Integration

2.1 Web Services Choreographies

WS-CDL 1.0 is, at this moment, a working draft of the W3C. Its purpose is the precise description of the collaboration among parties, independent from the respective implementation details. Verification is not a concern of this standard, although researchers are proposing different approaches [7] for enriching descriptive standards with verification artifacts.

Basically, WS-CDL describes a *choreography*: the collaboration pattern among the interacting parties, in terms of web services invocations, information interchanges, etc. Once this model is agreed upon, the involved organizations can develop their role independently, according to their implementation interests. As far as the "global contract" is honoured, the interoperability will be guaranteed.

2.2 Web Services Orchestrations

On the other hand, and in contrast, BPEL4WS allows the description of a business process in an unilateral way. It is possible to describe:

- Executable processes (BPEL4WS can be translated into an implementation)
- Abstract, non-executable processes

There are different opinions about the meaning of *choreography* and *orchestration*, but we can assume that the orchestration involves mainly the combination of existing web services to define a new one, whereas the choreography offers a global framework for the orchestrations to cooperate. The orchestration could be viewed as an individual, internal business process (Fig. 1).

3 Protocol Compatibility

3.1 Selection of the Theoretical Framework

Several formal models are available for verifying protocol compatibility. CSP or π-calculus are an interesting option [7]; in fact, there exists a close relation between WS-CDL and π-calculus, although this has not been formally established [8]. In spite of their robustness, formal methods in general have to face technology transfer problems for their adoption in everyday practice by the industry.

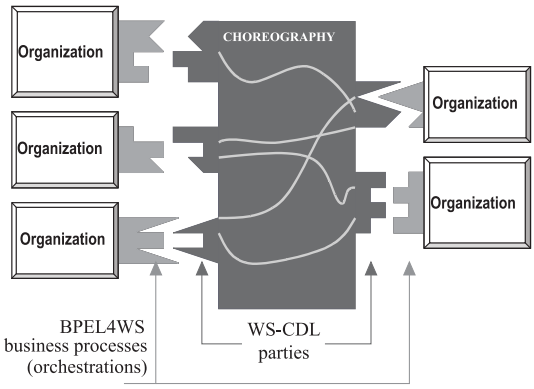

Fig. 1. Modelling web-service-based business processes by means of choreographies (WS-CDL) and orchestrations (BPEL4WS)

Our proposal, complementary to the initiatives based on formal methods, lies in the model proposed by Yellin and Strom [9]. These authors define a straightforward notation for specifying synchronous protocols, based on state machines. Given two protocol specifications, algorithms are described for detecting some kinds of defects. In previous research we have established that both the protocol description and the compatibility verification algorithms can be easily implemented over a CLP-based inference engine [1].

3.2 The Yellin and Strom Model

Yellin and Strom define a model (that we will call *YS model*) for complementing application interface specifications by means of sequence restrictions called *protocols*. A protocol makes explicit the relations among the application messages.

Their verification is focused on determining whether two given protocols are compatible in the sense that their interaction is free from certain kind of errors: *unspecified receptions* (that is to say, invocations that were not expected by the receiver) and *deadlocks*. For non-compatible protocols, Yellin and Strom describe how to automatically derive adaptors that make their interaction possible (this latter aspect is not addressed in the work presented here).

In the YS model collaborations always happen between two parts. This may seem a severe limitation for dealing with multipart business processes, but complex collaborations can be reduced to their bilateral components. And in our particular case, the goal is to study the compatibility between a party specification in the choreography and the business process defined by an orchestration, so there are always no more than two operands in each verification.

Following the YS model, a component exposes an interface for each interaction pattern, and such interface has a collaboration specification. This specification consists of the *interface signature* (the set of messages that can be sent

and/or received) and the *protocol* (the set of sequence restrictions denoted by means of a finite state grammar). The protocol is equivalent to a state machine in which any transition is the result of sending or receiving a message. There are no gratuituous locks, since a system is not allowed to stay indefinitely in a state where it can send a message (it will send it sooner or later, making a transition). The communication in the model is synchronous, because this makes reasoning about the systems easier.

4 From Specification to Verification

4.1 Fundamentals

The core of our proposal is to establish a correspondence between the control structures and interface declarations of the business models (both at choreography and orchestration levels), on one hand, and the YS model, on the other hand (that is to say, state machines). Then, Yellin and Strom theory can be applied to the verification of compatibility between each of the orchestrations (the particular business process of each party) and the global choreography (the common interaction agreement). Such correspondence seems feasible, since WS-CDL is mainly a declarative notation and BPEL4WS processes are basically the expression of an algorithm as a flow diagram [3]. Figure 2 shows this approach.

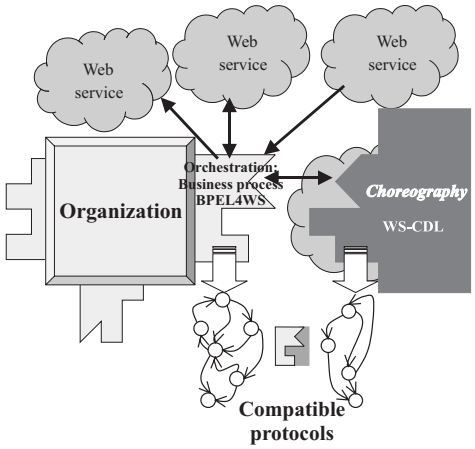

Fig. 2. Verifying compatibility between business processes and global choreography through protocols

After translating a WS-CDL specification into the YS model, other YS specifications for each party's behaviour will be taken (be it directly or as a result of extracting them from BPEL4WS specifications, depending of the most appropriate strategy for the particular enterprise context). Using these resulting

protocols, the YS algorithms can be used for verifying the compatibility between each business process and the defined choreography.

4.2 Choreography

At the most basic level, WS-CDL activities are organized following sequential, parallel and alternative structures; other higher-level constructions exist (such as *WorkUnit*). Sequential and alternative structures are not an issue, but the parallel one is a usual source of trouble when state machines are involved [10]. Expressing WS-CDL parallel structures in the YS model could lead to a state space explosion (some techniques have been proposed to address such problems [11]).

Having that into account, the first step would be to define a detailed correspondence between the different kinds of interactions in the specification [6] and the sent and received messages in YS terminology. Other activities of WS-CDL, such as *NoActivity*, are internal and do not have much influence in the procotol.

4.3 Orchestration

As for the orchestration, that is, the modelling of business processes of each involved entity, some details are worth mentioning. The general approach is similar to that of the choreographies. Leaving aside complex questions, like error handling, there are:

- Primitive activities (<`invoke`>, <`receive`>, <`reply`>) that can be identified with messages in the YS model.
- Activities which are not relevant for protocol compatibility, like <`empty`>.
- Structural activites, like <`sequence`>, <`switch`> or <`while`>, which may fit comfortably in the YS model.
- Other structural activities, like <`pick`> or <`flow`>, that can be troublesome for their description in the YS model.

The <`pick`> structure implies a non-deterministic transition. YS model does not provide a specific notation for this, but it seems that the incoming event that determines a <`pick`> transition in the orchestration could be represented as an incoming message in YS. The <`flow`> structure brings again the aforementioned problem with parallel execution; the same considerations exposed for choreographies essentially apply for orchestrations.

In the case of orchestrations, however, there is a specific question: as shown in Fig. 2, a BPEL4WS business process may invoke very different web services, outside the choreography definition. YS model, however, can only perform bilateral compatibility verifications. Hence, some procedure must be provided for dealing with messages that are not related to some of the two parties involved in a verification. YS model provides an interesting resource for solving this problem: it defines the concept of *subprotocol*, that can be compared to the notion of *subtypes* in type systems. A protocol P' is a suprotocol of P iff all of the following statements apply:

- The initial state of P' is the same of P
- Every final state of P is also a final state of P'
- P' can be obtained from P by applying the following operations:
  - Adding a set S of new states to P
  - Adding a set of new *receive* transitions to P
  - Adding a set of new *send* transitions to P', such that each new transition begins at one of the new states (the ones belonging to S)

It can be proved [9] that if P_1 is a subprotocol of P and P is compatible with P_2, then P_1 is compatible with P_2. The full business process will contain additional states, sendings and receptions related to web services that are not included in the choreography; but if it can be described as a subprotocol of a protocol wich is, in turn, compatible with the choreography (considering the interactions with foreign services as internal, irrelevant operations) the compatibility between the business process and the choreography will be ensured.

4.4 Verifiable Subset

So far, some limitations of the YS model as an intermediate representation means for BPEL4WS and WS-CDL have been remarked, and some possible ways for solving them have been pointed out. But a different yet appealing approach would be the definition of a *verifiable subset* of both notations.

Given any static analysis technology, it is usually not too difficult to formulate a counterexample which that technology cannot easily process, or which raises one of the typical decidibility problems. These counterexamples, however, might not be relevant from a practical, realistic point of view. It is usually acceptable to establish some limits in the use of programming languages or notations, as a tradeoff to achieve quality goals[1].

When dealing with notations for developing business processes, it may be also acceptable an initial approach in which some (even significant) limitations are self-imposed in exchange for automatic verification tools; these limitations can be removed as new verification techniques are developed. We propose an approach in which only a *verifiable subset* of both BPEL4WS and WS-CDL is used when an automatic verification system is desired.

5 Static Verification as the Foundation of a Design Methodology

5.1 Building Multiparty Business Processes

With the purpose of incorporating static analysis to the software development process, the previously defined verification process can be converted into a *construction* process. First, a verifiable subset of WS-CDL (as described in Sect. 4.4)

[1] Structured programming is an obvious example; the development of OWL-Lite because of decidibility problems in the full OWL is another.

will be used for the choreography definition. Once the choreography has been defined and agreed upon by the involved parties, the web services for the business processes will be developed by a *derivation* process.

5.2 Deriving Business Processes

The derivation of business processes can rely on another interesting property of the YS model. Given a protocol P, the *inverse protocol* $\overline{P}$ can be defined from P by inverting the direction of every message, so that sent messages become received messages and vice versa. Although this could not be ensured with asynchronous semantics, under the synchronous semantics of the YS model it is guaranteed that $\overline{P}$ *is always compatible with* P.

This property of te inverse protocol offers the starting point for a process for defining the business process of each party, given a choreography (Fig. 3). If a protocol P (in YS terms) is derived for a party of that choreograpy, the inverse protocol $\overline{P}$ may be used as the foundation for the design process (the construction of the inverse protocol is a well established, fully automatable process). This inverse protocol may then be altered by the designer by applying transforms to obtain a final protocol $\overline{P}'$, provided that the conditions exposed in section 4.3 apply; this will ensure that $\overline{P}'$ is a subprotocol of $\overline{P}$. Since a subprocotol is always compatible with its base procotol, and since the inverse protocol $\overline{P}$ is in turn proved to be compatible with the original protocol P, the result of this derivation is a protocol which, by construction, is compatible with the protocol denoting the behaviour of a party in the original choreography.

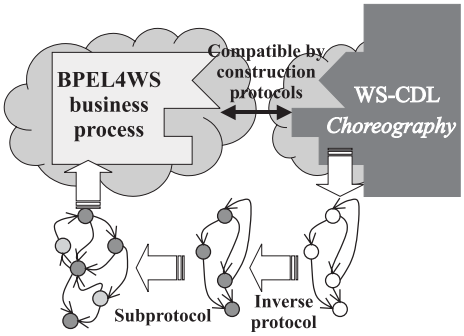

Fig. 3. A development method for designing business processes which are compatible, by construction, with the agreed choreography

If the transformation between protocols of the YS model and BPEL4WS orchestrations is well defined, a business process definition (optionally executable) can then be inferred from the final subprotocol. Following this procedure, it

would be possible to build a business process description in BPEL4WS which, by construction, fits the original contract defined by the choreography, guaranteeing at least that there are neither unspecified receptions nor deadlocks bewteen both sides of the collaboration.

6 Conclusions

It seems possible to develop a verification framework for WS-CDL and BPEL4WS offering additional guarantees about the robustness of the collaboration and its implementation, by means of widely available technologies. In this paper, an approach for verifying protocol-level compatibility (avoiding unspecified receptions and deadlocks), based on previous work by Yellin and Strom, has been presented.

The notions of inverse protocol and subprotocol of the YS model are the basis of a method for the reliable development of multipart business processes (choreographies) at the SOA level. The presented approach emphasizes the idea of incorporating static analysis technologies for the automatic verification of the work being done and early error detection.

For the practical application of these ideas, it seems desirable to define a *verifiable subset* of the notations involved; the automatic verification will probably come at the price of some loss of expressive power, but this is a usual tradeoff in software engineering when quality is prioritized.

References

1. Cernuda del Río, A.: Sistema de verificación de componentes software. PhD thesis, Servicio de Publicaciones (Universidad de Oviedo) (2002) ISBN: 84-8317-316-6.
2. Cernuda del Río, A., Labra Gayo, J.E., Cueva Lovelle, J.M.: A model for integrating knowledge into component-based software development. In: 4th International ICSC Symposium - Soft Computing and Intelligent Systems for Industry, University of Paisley, ICSC Academic Press (2001) ISBN: 3-906454-27-4.
3. Weerawana, S., Curbera, F.: Business process with bpel4ws: Understanding bpel4ws, part 1. IBM developerWorks (2002)
4. Khalaf, R.: Business process with bpel4ws: Learning bpel4ws, part 2. IBM developerWorks (2002)
5. Duftler, M., Khalaf, R.: Business process with bpel4ws: Learning bpel4ws, part 3. IBM developerWorks (2002)
6. : Web services choreography description language version 1.0. W3c working draft (2004)
7. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. (2005) To appear in Electronic Notes in Theoretical Computer Science.
8. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language (ws-cdl). BPTrends (2005)
9. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems **19** (1997) 292–333
10. Plasil, F., Visnovsky, S.: Behavior protocols for software components. IEEE Transactions on Software Engineering **28** (2002)
11. Visnovsky, S.: Modeling Software Components Using Behavior Protocols. PhD thesis, Charles University (Chzec Republic) (2002)

Cost-Effective Service Composition

Hakan Hacıgümüş

IBM Almaden Research Center, *hakanh@acm.org*

Abstract. Service Oriented Architectures (SOAs) provide a standard platform where applications and other IT resources are distributed and seamlessly accessible. In such an environment, a solution offering is composed from a set of primitive service components, which is called service composition. The selection of service compositions under the system constraints is a crucial issue both from the technical and the economic perspectives. In this paper, we investigate the problem of composing services to meet the multiple consumers' business goals while minimizing the overall delivery cost to the consumers. We model the problem as an optimization problem and present a formal solution.

1 Introduction

Service Oriented Architectures (SOAs) provide a standard platform where applications and other IT resources are distributed and seamlessly accessible. Although these applications and resources deliver significant value individually, a greater value can be derived by combining several services from different providers. In such an environment, a solution offering is composed from a set of primitive service components, which is called service composition. Many recent studies have focused on the service composition issues [2, 4, 1]

The service provider companies started to conceive innovative ways to deliver the services to the consumers. One of the fastest growing models is establishing service hubs that integrate heterogeneous service components to deliver solutions to consumers' business needs, which are typically modeled as business processes.

Along with the rich set of functionalities, the new computing paradigm promises economic incentives both for the service providers and the consumers. The multi-tenancy model allows sharing the standardized service compositions among the multitude of consumers with minimal or no customization thereby reducing the fixed-costs per consumer. The service integration model also allows the service providers and consumers to shop for the best mix of service components from an ever growing number of providers.

Under this scenario, the service provider receives the request from the consumers and provides integrated service solutions to satisfy the consumers' business needs. The consumer requests are defined by business processes and the integrated service solutions are created by composing heterogeneous service components. This model defines three main entities in the system; *business processes*, *service compositions*, and *service components*. A service component is essentially any functional unit that can be exposed as a service. This functional unit can be

a part of implementation or whole application. A business processes is a collection of related, structured activities that produce a specific service or product. The business goals can be defined as the outputs of the business processes. Naturally, the consumers want to match the services that they are receiving from the service providers with the internal business processes. We can define this association in terms of delivering the necessary functions in the form of services to complete the end-to-end lifecycle of all business processes defined in an organization.

Our goal is to find the service compositions that satisfy all of the consumer requests while minimizing the overall cost of service delivery across all of the consumers. It should be observed that the delivery costs of a service component could be different if it is delivered individually or as a part of a particular service composition. Because, packaging a set of service components may result in reduced fixed-costs for the whole package as compared to the sum of the delivery costs of the individual components. In addition, different service compositions might produce different cost schemes.

In the scope of this paper, we assume that there are existing service compositions available in the system. Our goal is to find the set of service compositions, which satisfies our objectives, from the given set of service compositions. The existing service compositions could be defined and constrained by many different sources. Some examples are existing product implementations, interoperability issues, and contractual agreements. This is an orthogonal problem to the main focus of this paper. In this work, we propose a model which is independent of any given set of service composition and application specifics. The model is defined at the abstract level and could take any set of given service compositions as an input.

The service composition issues have been studied in the literature. [6] presents a comparative study of service selection methods to form the optimal composition of candidate services. [5] discusses service composition in the service overlay networks. In that work, the optimization is formulated as a local selection strategy, meaning a component service is assigned to an individual task, one at a time. [7] proposes the use of fuzzy sets to express the users' QoS preferences that are used in service composition process. In [8], the authors study how to quickly compose web services when a large number candidate services are available. They present algorithms for fast membership checking to filter the candidate services that will be used in compositions afterwards. The work presented in [9] is closely related to our work. The authors argue that the selection of the components services should be carried out during the execution of a composite service, rather than at the design-time. The service selection is formulated as an optimization problem which can be solved using linear programming methods.

2 The System Model

In this section we present the system model in more detail. We consider the main system entities, namely, business processes, service compositions, and ser-

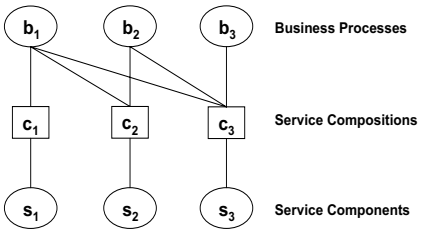

Fig. 1. Individual service components

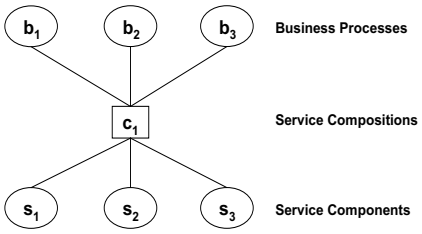

Fig. 2. Single service composition

vice components, and the optimization problem as described in the previous section. To illustrate the problem, consider the following example scenario. Assume that there are three consumers requests that need to be fulfilled and they are represented as the business processes: b_1, b_2, b_3. The service provider has three distinct service components, s_1, s_2, s_3, that can be delivered either individually or as a part of a service composition. Also assume, that the business process b_1 requires the services that are implemented by the service components s_1, s_2, s_3, b_2 requires s_2, s_3, and b_3 only requires s_3. There are individual costs associated with the service components s_1, s_2, s_3: 3,8, and 11, respectively. These costs represent the service delivery costs if the service components are delivered to the consumers individually.

The extreme case where all of the service components are delivered individually is shown in Figure 1. As the service components are delivered individually, the service composition nodes are identical to the service component nodes. An edge between a service component and a service composition shows that the service component is included in the service composition. An edge between a service composition and a business process shows that the business process receives some or all of the required services through the service composition. Then, the total cost for b_1 is $3 + 8 + 11 = 22$, for b_2 it is $8 + 11 = 19$, and for b_3 it is 11. The total service delivery cost for all of the business processes is 52.

Another extreme case is depicted in Figure 2, where there is only one service composition that integrates and delivers the system components together. The associated costs are different in this case. Assume that the integration cost for s_1, s_2, s_3 is 4. That means if they are packaged together and delivered as a service this is the cost added to the sum of individual costs. As the service components are not delivered individually their individual costs are also different; 2, 5, and 6 for s_1, s_2, s_3, respectively. Although b_2 and b_3 do not need all of the service components they are charged for the integrated service delivery cost as it is the only implementation in the system. In this case, the total cost for b_1 is $4 + 2 + 5 + 6 = 17$, which is obtained by adding the integration cost to the individual costs of the service components. The total costs for b_2 and b_3 are also the same as there is only one service composition is available in the system. Therefore the total service delivery cost for all of the business processes is 51.

Next, we consider another scenario that is shown in Figure 3. Here there are three service compositions c_1, c_2, c_3 available. c_1 directly delivers s_1, c_3 directly

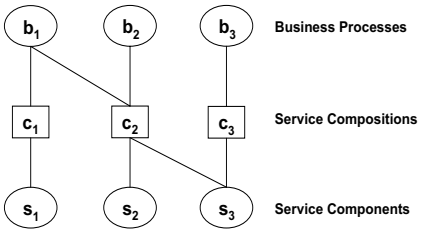

Fig. 3. Optimal service composition

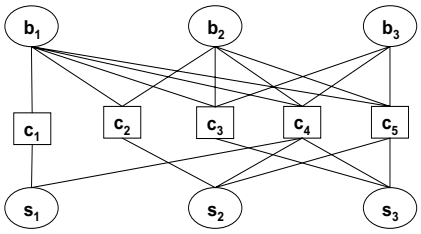

Fig. 4. All available service compositions

delivers s_3, and c_2 integrates s_2 and s_3. Assume that the integration cost for s_2 and s_3 is 3 and the individual and integrated costs are the same with the previous cases. Then, the total cost for b_1 is $3 + 3 + 5 + 6 = 17$, for b_2 it is $4 + 5 + 6 = 14$, and for b_3 it is 11. The total service delivery cost for all of the business processes is 42.

Figure 4 shows the system if we combine all of the cases described above. The system has five possible service compositions covering the given cases. As we stated before, our system model takes any given set of available service compositions defined by the system constraints. We define the optimization problem on top of this model definition.

The last case exploits a particular service composition scheme to reduce the total delivery cost to the consumers. If we define the system as in Figure 4, c_1, c_3, and c_5 would be chosen as the service compositions. b_1 would receive service from c_1 and c_5, b_2 would receive service from c_5, and b_3 would receive service from c_3. Note that these service combinations fulfill all of the consumer requests stated in the problem.

3 Problem Model

In this section, we formally present the **Optimal Service Composition Selection (OSCS)** problem in detail. As we stated above, our goal is to choose the most "beneficial" service compositions from the candidate service compositions while minimizing the overall cost to the consumers.

The *benefit* of a service composition is defined as the ability of the service composition to satisfy the service needs of the business processes, which is the essential goal of the system. This ability is represented by the connections between the business processes and the service components through the service components and formally modeled below. In the system we have three main entities: service components, service compositions, and business processes defined by the consumers.

We model the OSCS problem by using a graph model as depicted in Figure 4. In a *service delivery graph* $\mathcal{G} = (V, E)$, we have three types of nodes (vertices), S, C, B for service components, (candidate) service compositions, and business processes, respectively. Thus, $V = S \cup C \cup B$.

There is an edge between a service component node $s \in S$ and a service composition node $c \in C$ if s is included in c. We define an edge between a service composition node $c \in C$ and a business process node $b \in B$ if c is used by b, i.e., if business component b uses the part or all of the services provided by c to satisfy the consumer needs. As it is clear, there are no edges between the service component nodes and the business process nodes but they are *connected* through service composition nodes. Thus, $E = \{(u, v) \mid \text{either } (u \in S \text{ and } v \in C) \text{ or } (u \in C \text{ and } v \in B)\}$.

Definition 1. (Connected Pair) *$sb : s \in S, b \in B$ is called a* connected pair *$\iff \exists\ c \in C$ such that $(s, c) \in E$ and $(c, b) \in E$. We call this as c* connects *s and b, and denote it as $c \rightsquigarrow sb$. P is defined as the set of all connected pairs in the graph.*

Obviously, the system has to satisfy the consumer needs. Therefore, all of the business processes should be connected to the necessary services through the service compositions. In the model, this means, there should be at least one sb connected pair for each $b \in B$.

Definition 2. (Service Component Cost) *A cost s' is defined for each service component node s. This cost represents the cost to deliver the corresponding service component to the consumer individually.*

Definition 3. (Service Composition Cost) *A cost c' is defined for each service composition node c. The cost c' has two terms, 1) a fixed cost p that includes the system specific cost elements for the service composition, such as integration cost, delivery cost, etc., to deliver the service composition to the customer, and 2) the sum of the individual integration costs of the service components that the corresponding service composition is comprised of. Formally, $c' = p + \sum s''_i$, where s''_i is the integration cost for $s_i \in S$ and $\forall\ (s_i, c) \in E$.*

Note that s' and s'' costs are different. s' represents the cost to deliver the corresponding service component to the consumer individually. s'' is specific to the service composition that the service component is included in. s'' is the cost of the service component for the specific service composition.

Definition 4. (Measure) *We define measure $M(\cdot)$ for a graph $\mathcal{G} = (V, E)$ as $M(S, C, B) = \sum c'$ the sum of the service composition costs, where c' is the service composition cost for $c \in C$.*

Definition 5. (Optimal Service Composition Selection (OSCS) Problem) *We define the OSCS problem as for a given service delivery graph $\mathcal{G} = (V, E)$, find a subset $C' \subseteq C$, such that $M(S, C', B)$ is minimum over all possible $C' \subseteq C$.*

Now, we provide the definition for the Weighted Set-Covering problem, which we will use to prove the NP-hardness of OSCS problem.

Algorithm : *GREEDY-WEIGHTED-SET-COVER*
Input: A collection P of sets $Q_1, Q_2, \ldots, Q_n$ s.t. $U = \bigcup_{Q_i \in P} Q_i$ and associated costs $c_1, c_2, \ldots, c_n$
Output: The cover P' of minimum cost

```
1 U' ← U
2 P' ← ∅
3 while U' ≠ ∅
4   do select an Q_i ∈ P that maximizes |Q_i ∩ U'|/c_i
5   U' ← U' − Q_i
6   P' ← P' ∪ {Q_i}
7 endwhile
8 return P'
```

Fig. 5. Greedy Algorithm for weighted set-covering

Definition 6. (Weighted Set-Covering Problem) *An instance* (U, P) *of* Weighted Set-Covering *problem is defined as follows: A finite set of points* U*, a collection* $P = \{Q_1, Q_2, \ldots, Q_n\}$ *of subsets of* U*, and positive numbers, i.e., costs,* $t_1, t_2, \ldots, t_n$ *associated with* Q_i*. The cost of a collection is* $\sum_{Q_i \in P} t_i$*. A cover for* U *is a subset* $P' \subseteq P$ *such that every element in* U *belongs to at least one member of* P'*. The problem is to find a cover of minimum cost.*

The Weighted Minimum Set Cover problem is known to be NP-hard [3].

Theorem 1. *Optimal Service Composition Selection (OSCS) problem is NP-hard.*

Proof: We prove the theorem by a reduction from the Weighted Set-Covering problem. The detailed proof is omitted due to the space limitations. ∎

4 Greedy Approach

In this section we present an approximation algorithm, with a ratio bound, to the weighted set-covering problem. This will provide a basis for our implementation of a greedy algorithm to solve the OSCS problem. The approximation algorithm for the weighted set-covering problem is shown in Figure 5 [3].

The set U' contains the set of uncovered points at each stage of the algorithm. The set P' holds the cover that is being constructed. In Line 4, we select a subset Q_i that maximizes the ratio $|Q_i \cap U'|/c_i$, breaking ties arbitrarily. The ratio counts the number of points covered by Q_i per unit cost. After a Q_i is chosen, its elements are removed from U' and Q_i is added to P'. When the algorithm terminates, P' is the cover of minimum cost.

Theorem 2. *GREEDY-WEIGHTED-SET-COVER algorithm has a ratio bound of:* $|P'| \leqslant |P^*| \cdot H(max\{|Q_i| : Q_i \in P\}) \leqslant 1 + \ln|U|$*, where* $|P^*|$ *is the cost of*

Algorithm : *SERVICE-COMPOSITION-SELECT*
Input: C: set of candidate service compositions, associated with service composition costs c', S: set of service components, B: set of business processes, SB: set of all connected pairs sb
Output: C': set of selected service compositions

Precomputation: Construct a matrix $SC = (f_{ij})$: $1 \leq i \leq |S|, 1 \leq j \leq |C|$ such that $f_{ij} = 1$ if service component $s_i \in S$ is included in service composition $c_j \in C$, $f_{ij} = 0$ otherwise

Computation:

```
1  C' ← ∅, SB' ← ∅, SB'' ← SB
2  while SB'' ≠ ∅ do
3    C'' ← C − C',SB'' ← SB − SB'
4    for all c ∈ C'' let benefit(c) = 0
5      for all b ∈ B do
6        for all c ∈ C'' do
7          if c is included in b do
8            for all s ∈ S do
9              if f = 1, f element of SC corresponding to s and c
10               and no previous connection between b and c then
11                 benefit(c) ← benefit(c) + 1
12     endfor, endfor, endfor
13   select c_select ∈ C'' maximizing the ratio benefit(c)/cost(c)
14   C' = C' ∪ {c_select}, SB' = SB' ∪ {sb_i|c_select ⇝ sb_i}
15 endwhile
16 return C'
```

Fig. 6. Greedy algorithm service composition selection

an optimal cover P^* and $|P'|$ is the cost of the cover P' returned by the greedy heuristic algorithm.[1]

The proof for the theorem can be found in [3]. Based on the greedy heuristic, we implemented a greedy algorithm, which works on our graph model. Pseudo code for the algorithm is given in Figure 6. An adjacency matrix, SC, among the service component and the service composition nodes is constructed in the precomputation step. This matrix is utilized to avoid repeated comparisons to test the inclusion of a service component in a particular service composition in the loop starting at Line 8.

In each iteration, the benefits of the service composition nodes c in candidate service compositions set C'' are computed and the service composition that maximizes the ratio $benefit/cost$ is added to the selected service composition set C' (Line 14). Benefit for a service composition is computed as the number of previously uncovered connections, which are through the service composition node, between the service component nodes and the business process nodes. Selected service composition are removed from the candidate service compositions

[1] $H(\cdot)$ is a Harmonic number and nth Harmonic number is $\sum_{k=1}^{n}(1/k) = \ln n + O(1)$.

set in Line 3. We also keep updating the set of connected pairs SB of service components and business processes. When a service composition node is selected, all of the connected pairs that are created by that service composition node are also removed from the graph. (Line 3). This is required as the benefit of a service composition node is the number of connections that are made through the service composition node. Consequently, the benefit of each service composition node is changed over the iterations. The algorithm terminates when all of the connected pairs *sb* between the business processes and the service components are covered. This is the essential problem we solve, i.e., assigning the necessary service component to "all" of the business processes.

5 Conclusions

We have studied the problem of optimally selecting the service compositions. Our goal was to to meet the multiple consumers' business goals while minimizing the overall delivery cost to the consumers. We have formulated the problem as a graph problem and the service composition selection problem has been defined as an optimization problem. We have shown that the problem is NP-hard and presented an approximation algorithm with a ratio bound as a solution.

References

1. D. Berardi, D. Calvanese, G. Giacomo, M. Lezerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. of ICSOC*, 2003.
2. A. Charfi and M. Mazini. Hybrid Web Service Composition: Business Processes Meet Business Rules. In *Proc. of ICSOC*, 2004.
3. V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
4. Z. Duan, A. Bernstein, P. Lewis, and S. Lu. A Model for Abstract Process Specification, Verification and Composition. In *Proc. of ICSOC*, 2004.
5. X. Gu, K. Nahrstedt, R. Chang, and C. Ward. QoS-Assured Service Composition in managed Service Overlay Networks. In *Proc. of ICDCS*, 2003.
6. M. Jaeger, G. Muhl, and S. Golze. QoS-aware Compostition of Web Services: A Look at Selection Algorithms. In *Proc. of ICWS'05*, 2005.
7. M. Lin, J. Xie, H. Guo, and H. H. Wang. Solving QoS-Driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction. In *Proc. IEEE Intl. Conf. on e-Technology, e-Commerce and e-Service (EEE'05)*, 2005.
8. S. Oh, D. Lee, and S. Kumara. Flexible Web Services Discovery and Composition using SATPlan and A* Algorithms. In *Proc. of Modeling Decisions for Artificial Intelligence Conference*, 2005.
9. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality Driven Web Service Composition. In *Proc. of the 12th WWW Conference*, 2003.

Standardization approach for Federated ERP systems based on Web Services

Nico Brehm, Jorge Marx Gómez

Faculty of Computer Science, Otto-von-Guericke-Universität Magdeburg,
Universitätsplatz 2, 391o6 Magdeburg, Germany
{brehm, gomez}@iti.cs.uni-magdeburg.de

Abstract. *As enterprise resource planning (ERP) systems become more complex the financial expenditures that are connected to the application of such systems dramatically increase. ERP systems consist of many software components which provide specific functionality. However, these ERP systems are designed as an all-in-one solution, often implementing functionality not needed. Furthermore, such ERP systems depend on very large-scale infrastructures like servers and networking technology, which are very expensive to install and to maintain. The new idea is to develop a novel ERP system architecture which facilitates an overall reusability of individual business components (BC) through a shared and NON-monolithic architecture based on a peer-to-peer (P2P) network. In this paper we present a standardization approach which uses Web Services to wrap ERP components that are provided by a distributed system which appears as an ERP community.*

Keywords: ERP, Web Service, Web Service Standardization, WSDL, SOA, Peer-to-Peer

1 Introduction and motivation

Definition 1:
An *ERP system* is a standard software system which provides functionality to integrate and automate the business practices associated with the operations or production aspects of a company. The integration is based on a common data model for all system components and extents to more than one enterprise sectors [1, 2].

Modern ERP systems consist of many software components which are related to each other. Currently these components are administered on a central application server. In connection to the ERP system complexity several problems appear:

- Not all installed components are needed.
- High-end computer hardware is required.
- Customizing is expensive.

Due to the expensive proceedings of installation and maintenance only large enterprises can afford such complex ERP systems. One solution to face these problems is to develop a distributed ERP system where the system components are reachable over a network (e.g. internet). This component ensemble (federated system) still appears as single ERP system to the user, however it consists of different independent elements which exist on different computers. Based on this construction it is possible for an enterprise to access on-demand functionality (components) as services of other network members over a P2P network. This approach solves the mentioned problems as follows:

- Due to the separation of local and remote functions, no local resources are wasted for unnecessary components.
- Single components are executable on small computers.
- Due to decreasing complexity of the local system also installation and maintenance costs subside.

Definition 2:
A federated ERP system (FERP system) is an ERP system which consists of system components that are distributed within a computer network. The overall system functionality is provided by an ensemble of allied network nodes that all together appear as a single ERP system to the user. Different ERP system components can be developed by different vendors.

Definition 3:
An *ERP system component* in this case is a reusable, closed and marketable software module which provides services over a well-defined interface. These components can be combined with other components in a not foreseeable manner [4, p. 19].

2 Resource sharing in an ERP network

With the distribution of the ERP system components on different servers some advantages arise for the operators of each of these servers, because hardware demands are made to only their part of the total ERP system. This reduction of complexity facilitates administration and availability securing because all measures are confined to only one ERP peer whereas conventional ERP system operation is geared to the provision of all components at the same time. Figure 2 shows these two approaches in comparison to each other. The left hand side represents the architecture of a conventional ERP system where a closed amount of ERP components (C1, C2, …, C8) are installed on the same application server. The right hand side shows an open ERP network where each node is assigned to one ERP component which is provided as service (S1, S2, …, Sn). This P2P network consists of allied network nodes that all together represent a federated ERP system. New components are added as new ERP peers that provide corresponding services.

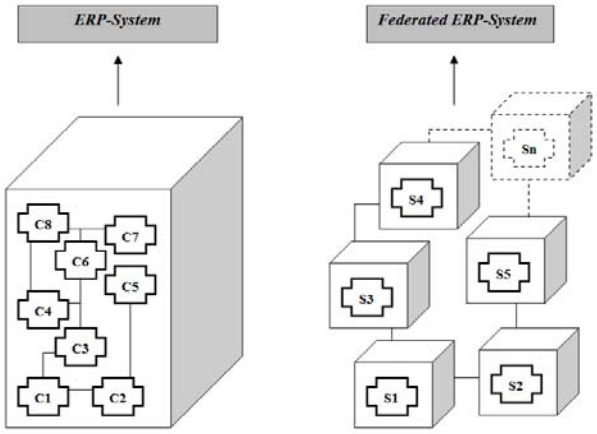

Figure 1: Conventional ERP system with ERP components (C) versus a federated ERP system that provides its ERP components as services (S)

As explained above, the distribution of the ERP system is based on a P2P architecture. Each peer can communicate with all the rest of the participating network nodes. The duties and responsibilities of every network node are divided into two sections. On one hand the service providing peers and on the other hand the ones which utilize theses services establish the basis for exchanging software components, whereas the overall system functionality will be available to the whole ERP network. The total charges and costs of the ERP system are averaged to all ERP network nodes, which means that each node saves expenses. This leads to the conclusion that the proposed component distribution to several servers has three main advantages:

- Every ERP server stores only a part of the total system.
- The open ERP network architecture allows the provision of more ERP components than only one server does.
- New ERP system versions arise by adding new components and peers to the network. New versions are available at once.

Perspectively the proposed distributed architecture paves the way to an *overall ERP system* that includes all imaginable functionality whereas the functions are provided as Web services. Although there are differences between Web services and business components a lot of similarities exist that argue for an equalisation of both technologies [3, p.1 ff]. Solutions and models for

- standardising,
- development,
- customisation
- and composition

of business components [4] must be given if we assume that the encapsulation of ERP functions as business components equals the provision of these functions as superior Web Services.

3 Example business process

Below we chose a manufacturing company to exemplify the distribution of ERP system functions. Figure 2 shows a typical example of a manufacturing business process as Event Process Cain (EPC) and in connection to this an opportunity of how internal functions can be processed by other ERP peers.

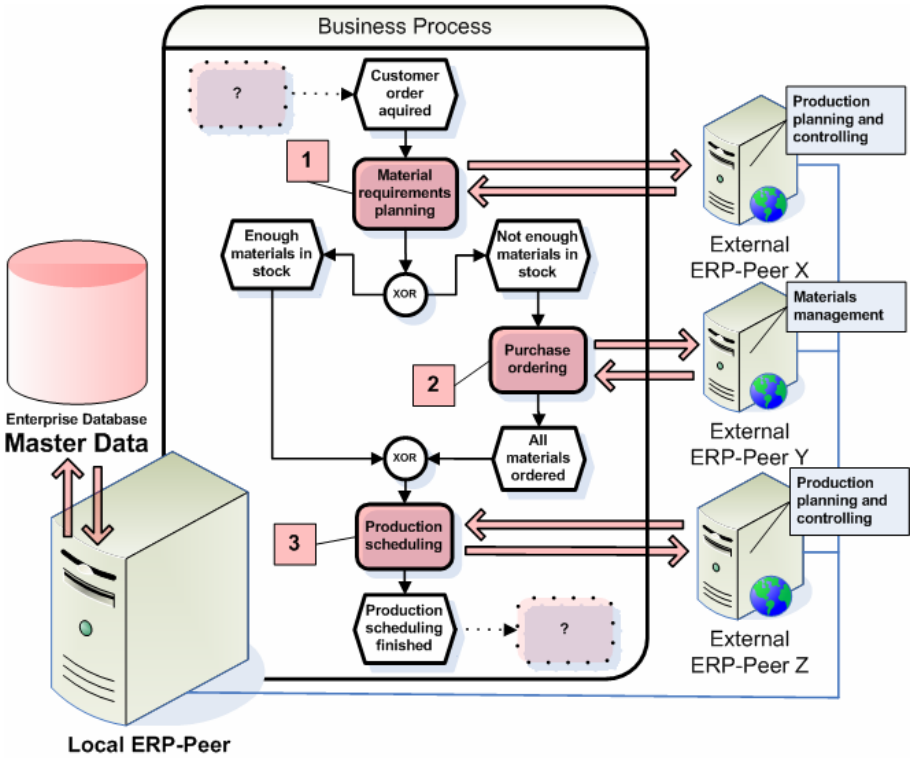

Figure 2: Example manufacturing business process and outsourcing of ERP-internal functions

The example above shows a highly simplified work process of a manufacturing company starting with the acceptance of the customer order until the completion of the production scheduling. The process contains three functions which are each supposed to be executed by external nodes of the P2P network.

- *Function 1*: After a customer placed a production order, it hast to be examined whether there is sufficient material in stock. The necessary input data which must

be sent to the external network node includes all Bills of Material (BOM) of the product to be produced and information about the stock of inventory as regards all individual parts. On this basis the function calculates whether production scheduling can start directly or whether anymore material has to be ordered at first. Finally the external function returns a respective information message back to the local business process.

- *Function 2*: If there is not enough material in stock the second function determines the missing material and sends material orders to the corresponding suppliers. The input values are BOM, stocks of inventory and supplier information. After the demand of material is calculated material orders are sent to the suppliers automatically. For simplification purposes we assume that the suppliers will deliver all materials in time without considering possible problems in this context. The return value of this external function is complex object which contains all executed material orders and the related delivery times.
- *Function 3*: When there is enough material in stock production is supposed to start. Therefore the third function creates all needed data records, e.g. optimized time schedules and work plans, and wraps them as a complex object which will be returned back to the requesting ERP peer. Among other things BOM, work places, capacities and production calendar data are required as input values of this function.

The local peer in this example wraps the required data as business objects which are included in the remote function calls. Complex return objects contain secondary master data records that are inserted into the local enterprise database. The example shows how single functions of a business process are executed on external ERP peers and how a local ERP system benefits from using external business logic, e.g. by using optimisation algorithms.

4 Standardization approach

Within the scientific environment of business informatics there is a variety of models existing which specify the different components of ERP systems in theory. The determination of an appropriate technology is the basis for a subsequent implementation. This proposal uses Web Services as communication basis and the exchange of messages respectively. The standardizing process focuses on the unification of ERP specific entity types, messages and functions. The following model refers to the description of an FERP standard as XML Schema.

Components of an ERP System

Before the elements of the FERP standard can be specified the functional structure of the target ERP system has to be fixed. The problem thereby is the amount of functions an ERP system can contain in principle. The following list gives a survey of the

different enterprise sectors where functions and processes of ERP systems can be assigned to. The list shows grouping categories for business objects in SAP/R3:

- Finance
- Treasury
- Controlling
- Investment management
- Enterprise controlling
- Real estate management
- Logistics
- Sales
- Logistics Execution
- Quality management
- Maintenance
- Customer service
- Project management
- Environmental management
- Human resource management
- Payroll accounting
- Event management
- Production planning and controlling (PP)
- Materials management (MM)
- …

Syntactic ERP component specification model

The basis of the ERP component specification model is the summarization of all functions that belong to the same sector of the functional business organization. Functions of the same sector are assigned to the same ERP component. The standardization of components includes the following levels:

- Syntactic level
- Behavioral level
- Synchronization level
- Quality-of-Service level

The following model describes the syntax standardization approach of ERP components and is based on the summarization of allied functions in components. This model defines three steps which are shown in figure 3:

1. Description of data types as XML Schema: All entity types of the component are specified in this document. In this manner the data model is described.
2. Description of message types as XML Schema: This document describes the request and response message types.

3. Description of function types as XML Schema: This document summarizes all functions of the ERP component to be standardized. On one hand the overall functionality of the component is framed and on the other hand the message types are assigned to the respective functions as input and output messages.

Based on this proposal it is possible that component providers cover an incomplete functional range because not all functions have to be mapped to a Web Services description. The idea in this case is that multiple vendors can contribute one part of the whole ERP component implementation. Likewise redundant functions are also possible.

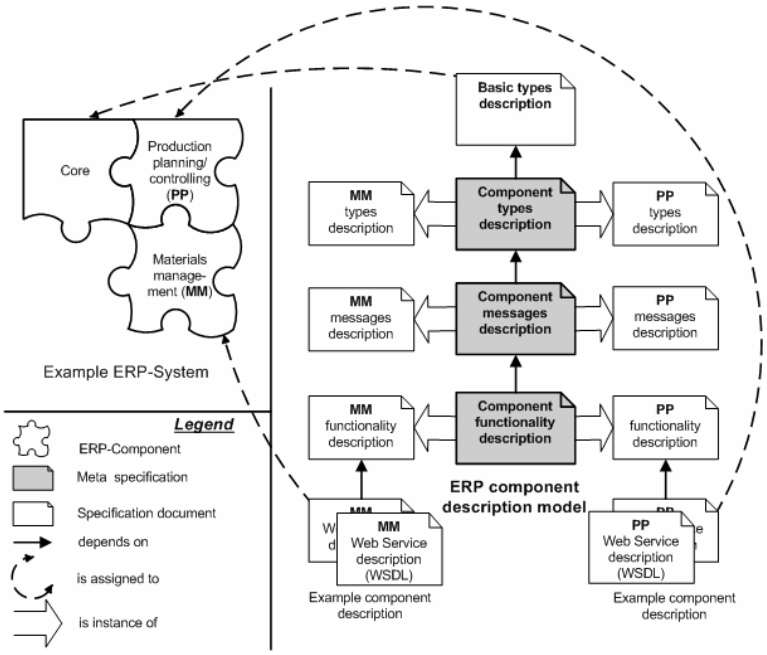

Figure 3: Syntactic ERP component specification model with example descriptions

Example specification

Figure 4 shows an uncompleted example specification of a production planning and controlling (PP) component. Based on the specification of data types e.g. BOM, work plan, operation etc. message types are specified. These message types are used to describe the functions of the PP-component. Out of this function specification different Web Services can be derived which implement the overall component functionality in a different completeness. The Web Service descriptions are expressed in the Web Service Description Language (WSDL) [5].

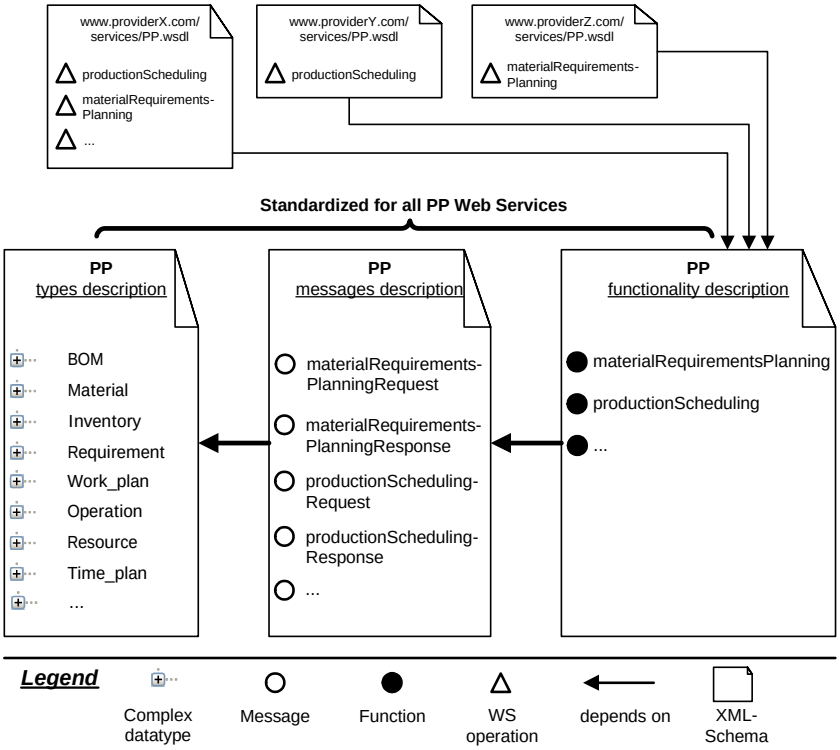

Figure 4: Uncompleted example component specification and three possible Web Service descriptions

5 Related works

Semantic Web Services are Web Services whose properties, capabilities, interfaces, and effects are encoded in an unambiguous, machine-understandable form [6]. Research works that deal with Semantic Web Services aim at the provision of a comprehensive Web Service description, discovery and mediation framework. Because UDDI, WSDL and SOAP are not sufficient other specification languages like Web Ontology Language (OWL) [7] and the Resource Description Framework (RDF) [8] are used. This approach is more abstract then the presented proposal because the efforts within the scope of Semantic Web Services do not concentrate on a specific type of Web Service. Our proposal is based on XML Schema to define a fixed language for ERP systems as basis for the communication between software components of a distributed application.

6 Conclusions and Outlook

Comparing distributed ERP systems and ERP systems running on only one computer, the distributed systems offer a lot of advantages. Particularly small- and medium sized Enterprises (SMB) benefit from using shared resources. However, the design and development of a system architecture is subject to a number of problems. The paper presents a basis for the standardizing process of ERP system components that are provided as Web Services. A standardized data model builds the basis for message and functionality specification.

The standardization of the syntactic level is only the first step. Behaviour, synchronization and quality of Web Services must flow into the definition of an overall ERP system standard. The future work must pick up these problems to realize the vision of a loosely coupled ERP system which allows the combination of software components of different providers.

References

1. Robey, D.; Ross, J.; and Boudreau, M. (2002): Learning to implement enterprise systems: An exploratory study of the dialectics of change. Journal of Management Information Systems, 19, 1, 17--46.
2. Rautenstrauch, C.; Schulze, T. (2003): Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker, Berlin et al.
3. Krammer, A., Turowski, K. (2001): Spezifikationabedarf von Web-Services. In: Ortner, E., Overhage, S. (Hrsg.): 1. Workshop „Entwicklung von Anwendungen auf der Basis der XML Web-Service Techologie. Darmstadt
4. Turowski K. (2003): Fachkomponenten: Komponentenbasierte betriebliche Anwendungssysteme, Aachen
5. W3C (2001): Web Services Description Language (WSDL) 1.1, URL: http://www.w3.org/TR/wsdl
6. S. McIlraith, T. Son, and H. Zeng (2001). Semantic Web services. In IEEE Intelligent Systems (Special Issue on the Semantic Web)
7. W3C (2004): OWL Web Ontology Language, URL: http://www.w3.org/TR/owl-features/
8. W3C (2004): RDF Vocabulary Description Language 1.0: RDF Schema, URL: http://www.w3.org/TR/owl-features/

Author Index