Bulletin of the Technical Committee on

Data Engineering

September 2008 Vol. 31 No. 3 **IEEE Computer Society**

Letters

Special Issue on Semantic Web Services: Composition and Analysis

Conference and Journal Notices

The TC on Data Engineering
Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin
The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

International Conference on Data Engineering

ICDE (the International Conference on Data Engineering) is the flagship database conference of the IEEE. The 2009 ICDE will be held in Shanghai, China at the end of March. I would encourage readers to check the "Call for Participation" on the back inside cover of this issue of the Bulletin for more details. ICDE has become not only one of the best database conferences, but one of the largest as well. I attend this conference every year and always find my time well spent. Not only is the research program first-rate, but there is an industrial program, demos, and workshops as well.

The organization exercising oversight for ICDE within the IEEE Computer Society is the ICDE Steering Committee. This organization has on-going responsibility for selecting the conference committees for each of the individual conferences, and for establishing the policies and procedures under which these annual conference committees operate. The outstanding success of ICDE during this decade has been enabled by the planning and management of this committee. Erich Neuhold was Steering Committee Chair until 2007. Since then, Calton Pu has been the Chair. More information about the Steering Committee can be found at `http://tab.computer.org/tcde/icde_stc.html`. Much is owed to Thomas Risse, who established the web site, and has done so much over the years, both for the Steering Committee and for the Technical Committee on Data Engineering, which sponsors both ICDE and this Bulletin.

The Current Issue

Not so long ago, transaction processing was considered to be a core part of the database field. Roughly speaking, transaction processing involves not only the database, but also applications written that exploit databases and transactions to provide functions that are specific to some particular problem area, e.g. business data processing. More recently, the database field, as measured by conference participation, has been moving away from transaction processing, perhaps considering it to be a solved problem. However, with the emergence of the web, and specifically web services, there are new solutions and new problems to be solved. The subject of the current issue of the Bulletin is "Semantic Web Services: Composition and Analysis". I hope that it will stimulate some further involvement by the database community in the area of applications exploiting databases, now in the new context of the web, cloud computing, and widely deployed applications.

The editor of the current issue is Jianwen Su. Jianwen has actively participated in the area of web services research and has focused this issue on this area. The resulting issue contains papers on a wide selection of topics dealing with web services: design, composition, reliability, analysis, reasoning about them, and more. What I like about the issue, and part of what the Bulletin is all about, is that the issue brings these papers together in one place, creating a very useful reference for further work as well as providing a great way to become familiar with this area. I want to thank Jianwen for his very successful effort in bringing this very interesting issue together.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

One of the fundamental changes the Web will bring to us is the ability to organize, manage, discover, compose, and invoke *web services*. Speaking loosely, web services are bite-sized pieces of software system components that can be executed over a network (e.g., the Internet). An immediate and laudable goal is to be able to dynamically share web services in a similar manner to the sharing of data on the Web. To this end, the principles of Service Oriented Architecture (SOA) offer useful guidelines for constructing web services, compositions, and applications based on web services.

Research on web services started more than a half dozen years ago and, in recent years, has attracted increasingly more attention from several research communities including the database community. It frequently reminds me of the data management research prior to the arrival of the relational data model. Back then, there was not only a lack of a simple, yet commonly accepted data model, but also the prevailing models (network, hierarchical) were very much motivated by physical storage structures for data. Indeed, the search for a suitable common model for web services is still ongoing in spite of various standards (e.g., the ones with names matching "$*$WS$*$"). An important reason for this is insufficient understanding about web services, their description, use, composition, management, their interaction with stored data, etc. (SQL standardization happened years after significant research efforts were spent on the relational data model, query languages, query optimization, database design, etc.) Of course, history doesn't quite repeat itself. Models for web services emerge from the physical world (software development), as well as from many different research communities with different twists, adding abstractions, semantics, data, orchestration/choreography, etc., or their combinations.

Many research challenges are in front of us. Here are three examples. Automation, as a key aspect of SOA, requires incorporation of *semantics* in various phases of web service development and applications. Providing semantics in suitable form is the first challenge. The standard SAWSDL enables the capture of semantics about how the input and output arguments of a web service are linked to underlying domain ontologies. However, research on the semantics of the actual impact of services and how they are combined (e.g., OWL-S, WSMO, FLOWS) continues to be an area of active investigation. A machine-readable semantics that can be reasoned about can ensure *correctness* and at the same time help to achieve *efficient* services. Service composition is another area full of interesting research problems. There are many possible models for composition. In one extreme, we might write down a detailed *orchestration* of a collection of web services. In the other extreme, we could start from a global *choreography* that defines the expected behavior of the entire composition. Between the two extremes, one could figure out behavior interfaces of two or more services (e.g., as "embedded" or partial choreography), and gradually expand to an entire composition. The exact relationship among the composition models remains to be explored. As the third example of challenging areas, techniques for analyzing, verifying, and mining web services are in high demand. These will become especially important as the services used are increasingly created through semi- and fully-automated discovery and composition mechanisms.

Presented in this special issue is a sampler of ongoing research efforts in web services. By no means do they represent a complete survey of current projects in the area, but I hope they can be the starting point of your exploration in web services. As you will find, there are many practical motivations for this work, but no lack of technical depth and elegance. I recall that Tim Berners-Lee in his WWW '03 invited talk identified data and (web) services as two fundamental elements flowing on the Web, interacting and bearing "fruits", freely and effectively. In my view, most of the work to date treats the service/process aspect and the data aspect in largely separated ways; the fundamental interplay of data and services, and the development of new models that bring that interplay to the forefront, remain largely unexplored and untapped. Maybe some of you will find deep and engaging challenges for your skills and talents by contributing to this exciting (and highly profitable!) grand challenge in software development and service management.

Jianwen Su
University of California, Santa Barbara

A Short Overview of FLOWS:
A First-Order Logic Ontology for Web Services*

Michael Grüninger
University of Toronto
Toronto, ON, Canada

Richard Hull
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA

Sheila A. McIlraith
University of Toronto
Toronto, ON, Canada

Abstract

FLOWS is a first-order logic ontology for Web services and a W3C Submission. In this article, we describe some of the motivation behind the development of FLOWS, together with its key features.

1 Introduction

The *First-order Logic Ontology for Web Services* (FLOWS) [9], also known as the Semantic Web Services Ontology (SWSO), was initially developed during 2002 to 2004 by a team of academic and industrial researchers, as part of the larger Semantic Web Services Framework (SWSF) effort, which culminated in a W3C Member Submission [10] in 2005. FLOWS was created on the premise that an unambiguous, computer-interpretable description of the process model of Web services and how they are composed, and client constraints on the services to be provided, are critical to automating a diversity of tasks, including Web service discovery, invocation, composition, monitoring, verification and simulation. To this end, FLOWS is based on first-order logic, and provides a rigorous axiomatization that captures the salient process-level semantics of Web services.

FLOWS is an extension of the Process Specification Language (PSL) [4], a first-order logic ontology for modeling processes that was originally developed for manufacturing processes, and realized as an ISO standard in 2004. FLOWS enables partial and/or complete specifications of the properties of Web services, including pre- and post-conditions, internal structures, composition patterns, messaging behaviors, and impact on the external world, all in the context of a rigorously axiomatized first-order logic framework. This article provides a very brief overview of FLOWS, including comparisons with related work, key principles underlying the creation of FLOWS, and illustrations of key components of the framework.

Background. A Web service *process model* describes the program that implements a Web service, which might itself be formed as a composition of other Web services. Over the years, a number of languages have been proposed for describing the process models of Web services. Some of the most important examples include Microsoft's XLANG, a Web service process modeling language based on pi-calculus; IBM's WSFL based on Petri Nets; BPEL4WS, a Microsoft, IBM, BEA, SAP and Siebel effort, which merges XLANG and WSFL; HP's Web Service Conversation Language (WSCL); BEA, Intalio, SAP and Sun's Web Service Choreography

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported in part by NSF grants IIS-0415195 and CNS-0613998, and by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Research and Innovation.

Interface (WSCI); BPML, backed by the Business Process management initiative; the XML Process Description Language (XPDL) backed by the Workflow Management Coalition; the Business Process Specification Schema (BPSS) of ebXML; and the W3C Choreography effort, WS-CDL, which draws on pi-calculus. The most popular language for Web Service orchestration in use today is WS-BPEL 2.0 (or BPEL for short), which offers significant enhancements over its predecessor, BPEL4WS.

In evaluating and comparing these efforts, a key observation is that they were designed to address a diversity of targeted process management tasks. Some, like BPEL, were designed to address Web service orchestration issues and to standardize workflow and execution with the objective of increasing transaction reliability and synchronization. Others, like WS-CDL, have focused on issues of Web service choreography, which involves message exchange to coordinate the activities of independent agents. As a consequence of the diversity of uses for which these languages have been designed, comparing them based on concept coverage is important, but not necessarily pertinent, as many of these languages could be extended to incorporate further concept descriptions.

It is our view that the most important shortcoming of these languages, and the one that is least easily addressed, is their lack of well-defined semantics. For example, several attempts have been made to formalize predecessors to WS-BPEL 2.0 using Petri nets, process algebras, and abstract state machines. While the WS-BPEL 2.0 specification is more precise with respect to the semantics, it is still informally defined [6].

In 2001, a coalition of semantic Web researchers, under the auspices of the DARPA DAML program, undertook to develop an ontology for Web services, using the Semantic Web ontology language DAML+OIL. This culminated in the creation of OWL-S (formerly DAML-S) [2] a Web service ontology developed in OWL (the successor of DAML+OIL) [5], and a W3C Member Submission in 2004. OWL is a family of knowledge representation languages for authoring ontologies and is endorsed by the World Wide Web Consortium (W3C). The semantics of most of the OWL languages, specifically OWL DL and OWL Lite, is based on artificial intelligence description logic, a subset of first-order logic. OWL Full (soon to be replaced by OWL 1.1) is based on a novel semantics that provides compatibility with RDF Schema. Most importantly OWL, and thus OWL-S, has a well-defined (formal) semantics, in constrast to efforts listed above.

Unfortunately, OWL has not proven sufficiently expressive to characterize Web service process models. While OWL-S does indeed have a description of the process model of a Web service, OWL is not sufficiently expressive to denote all and only the *intended interpretations* of that process model. As such, like other process modeling languages, the OWL process model must be human interpretted to resolve ambiguities, or translated to another, richer language, in which this new model can be unambiguously interpretted by a program. Indeed there have been four efforts towards defining the intended interpretation of the OWL-S (or DAML-S) process model: a Petri Net-based operational semantics [7], an interleaving function-based operational semantics based on subtype polymorphism [1], a semantics via translation to the first-order language of the situation calculus [7], and a semantics provided by translation to PSL [3].

OWL-S has many strong features. In particular, the concept coverage of OWL-S provides a firm foundation for process modeling efforts, including the ability to create both partial and complete specifications of relevant aspects of a group of Web services. Further, OWL's expressiveness limitations, which OWL-S inherits, exist to address the important trade-off between expressiveness on the one hand and decidability and tractability on the other, and are thus easily defensible in this context. Nevertheless, it was experience with OWL-S that, in part, motivated the development of FLOWS.

The Web Service Modeling Ontology (WSMO) [11] also provides an expressive, layered ontology for web services and their compositions, using a semantics based on a combination of description logic and horn logic.

Principles underlying FLOWS. FLOWS was developed based on the following three principles.

Provide a fully expressive language and framework. The goal of FLOWS is to enable reasoning about the semantics underlying Web services, and how they interact with each other and with the "real world". FLOWS does not strive for a complete representation of Web services, but rather for an abstract model that is faithful to the semantic aspects of service behavior. In that context, FLOWS enables a variety of reasoning tasks, by supporting descriptions of Web services that enable automated discovery, composition, and verification. This also includes the creation of declarative descriptions of a Web service, that can be mapped (automatically or through a systematic, partially manual process) to executable specifications. In particular, then, unlike the industrial process modeling languages listed above, FLOWS is intended to support, within one language and underlying framework, reasoning about Web services from a very broad range of perspectives.

FLOWS is a modular and extensible ontology. It is thus possible to provide alternative extensions to represent different approaches to message handling, choreography, and orchestration. As such, FLOWS can serve as an interlingua ontology that can facilitate interoperability of Web services that use different ontologies.

Use first-order logic as the basis. First-order logic enables the characterization of reasoning tasks for semantic Web services in terms of classical notions of deduction and consistency. FLOWS can be used to specify tasks in support of Web service discovery and composition; checking service properties such as reachability, liveness, and compliance with behavioral patterns and constraints; and querying about a wide range of semantic and temporal properties of services. This enables exploitation of off-the-shelf systems such as existing FOL reasoning engines and database query engines, thereby facilitating implementation and improving our understanding of the reasoning tasks. At the same time, the use of first-order logic does not preclude the use of alternative reasoning methods on selected subsets of FLOWS.

First-order logic has been criticized because it is semi-decidable (as opposed to OWL DL, which is decidable). Nevertheless, the motivating scenarios for semantic Web services show that in general we will need to solve intractable reasoning problems. Intractable reasoning problems are inherently intractable – using a different language does not make them tractable. The restriction to a language that is tractable simply means that there will exist reasoning problems that cannot be specified in the language.

Capture full semantics using an extensible family of axioms. Although other approaches to semantic Web services specify concepts contained in FLOWS, they do not provide a rigorous and complete axiomatization to support automated reasoning about the concepts. Incomplete axiomatizations require the use of additional extralogical mechanisms rather than reasoning from the axioms alone. Since automated Web services can share axioms, but not the specification of special-purpose axioms, incomplete axiomatizations restrict the reusability and sharability of an ontology.

In FLOWS, the process model for Web services and their compositions is formally specified using PSL This provides predicates and axioms that enable representation of, and reasoning about, core process modeling concepts, including fluents (that is, first-order predicates representing some portion of the "real world" that can change over time), activities (such as Web services), activity-occurrences (such as individual executions of Web services), and the values of fluents before and after activity-occurrences. The PSL standard is comprised of a layered collection of families of axioms that can be used to reason about a broad class of processes. As discussed in more detail below, FLOWS provides additional families of axioms, layered on top of a subset of the PSL axiom families, to enable representation of, and reasoning about, Web services and their compositons.

2 Key aspects of FLOWS

After providing some motivating use cases, this section briefly highlights some key aspects of the FLOWS language and framework, and illustrates how it can support the use cases.

Motivating use cases. The space of use cases that motivates and illustrates the need for computer-interpretable Web service process models is vast, ranging from discovery and composition to analysis, monitoring and error-recovery, and including activities such as queries about individual services and over families of services, contract enforcement, service histories, and provenance. We illustrate now with just a few examples, centered around services that focus on selling and shipping books.

1. *Inquiries about one service:* Does the Acme bookseller service always return a list of available second-hand copies of my requested book, if the book is out of print? Under what conditions does the Acme bookseller service permit me to pay for books using Paypal?

2. *Discovering services:* Find all bookselling services that will, at least in some cases, return a list of available second-hand copies of my requested book if that book is out of print. Or, find all bookselling services that will always return a list of available second-hand copies of my requested book, if it is out of print.

3. *Discovering composite services:* Find all book seller-shipper partnerships that are able to split shipments (e.g., as a consequence of delayed availability of some books) without an additional charge.

4. *Requesting (composite) services:* Create a service that can book a flight to Toronto, find and reserve a hotel room there for next Monday to Wednesday, identify the best way to get from the airport to the hotel, and ship a guide-book about Toronto to me at that hotel in time for my arrival. Furthermore, the hotel must be within 15 minutes travel (by foot and/or public transportation) of the Computer Science department.

5. *Responding to exceptions:* If the preceding scenario is underway and the flight into Toronto is delayed, then dynamically provide new recommendations on the best way to get from airport to hotel.

Ontology. As noted above, in the FLOWS ontology Web services (both atomic and composite) are represented as PSL *activities*, and Web service executions are represented as PSL *activity-occurrences*. Predicates that change in the real world due to activity-occurrences are modeled using *fluents*. An activity-occurrence is a limited, temporally extended piece of the world, with a clear temporal start-point and end-point.

As in PSL, the set of possible executions of a composite Web service is modeled essentially as a tree, whose nodes correspond to individual activity-occurrences, i.e., service executions, and where the children of one activity-occurrence correspond to the set of all possible activity-occurrences that could immediately follow it. For a fluent such as *book_available*(t, w) for title t and warehouse w, and service-occurrence o, the value of the fluent immediately before o occurs is given by the predicate *pre*(*book_available*(t, w),o), and the value of the fluent just after o occurs is given by the predicate *holds*(*book_available*(t, w),o). It is from these basic constructs that the full family of possible executions of a composite Web service can be represented and reasoned about. These constructs can also be used to specify pre-conditions and effects of services.

FLOWS also provides constructs for modeling the internal processing of composite Web services, including sequences, nondeterminism (i.e. alternative activities), iterated activities, conditional activities, and concurrency. As with Golog [8], these constructs are formally modeled as constraints, which enables both partial and complete specifications of processing characteristics. Activities can be decomposed into primitive activities or composed into more complex activities, and different classes of activities are defined with respect to ordering and temporal constraints on the subactivities. In this way, FLOWS supports reasoning with both complete and incomplete process specifications. In addition, FLOWS refines aspects of PSL with Web service-specific concepts and extensions, such as providing the infrastructure for representing messages between services.

Axiomatization. The FLOWS axiomatization is layered on top of the axiomatization of PSL Outer Core. The axioms provide a rigorous and complete specification of the FLOWS constructs, including such concepts as service, service-occurrence, messages and channels, control constructs for service composition, various kinds of constraints, and exceptions.

Enabling the use cases through queries and reasoning. The approach to support reasoning tasks with the FLOWS ontology and axioms is now illustrated with the use cases. The focus here is to illustrate the expressive power of FLOWS. It is clear that many of the problems that can be specified in FLOWS have high complexity or are undecidable; as noted above it is possible to create restricted versions of these reasoning tasks in order to obtain decidability and lower complexity.

Inquiries about an individual service, such as Acme book seller, can be achieved by reasoning over the tree of possible executions of the service. Note that both universal and existential quantification will be called for. For service discovery, the properties characterizing the desired services can be specified using formulas over the FLOWS ontology. This essentially reduces discovery to querying a database of service specifications. Even the composition of services is achieved through the specification of a formula with one free variable which describes the desired properties of the composition; each solution for this variable will be a (composite) service that provides a composition with the desired capabilities. Finally, FLOWS is able to represent the state of the world when a composite service has partially completed its execution. As such, it enables exception handling.

3 Closing Remarks

In this article we described some of the motivation behind the development of FLOWS, together with key aspects of the FLOWS ontology. In doing so, we argued that existing languages for modeling Web services were either lacking in expressivity, or did not have a well-defined semantics. As such, their ability to model and enable automated reasoning about Web services was limited. In contrast, FLOWS' use of first-order logic provides sufficient expressivity, a well-defined semantics, and a diversity of automated reasoning tools. FLOWS presents a natural evolution in the modeling of semantic Web services, reflecting a trend in semantic Web techonologies towards the use of more expressive ontology languages. Recent extensions to OWL, both realized and proposed, bear witness to this trend. Readers interested in further details on FLOWS are encouraged to consult the FLOWS (a.k.a. SWSO) specification at [9], which includes the full ontology and selected use cases.

References

[1] A. Ankolekar, F. Huch, and Katia Sycara. Concurrent execution semantics for DAML-S with subtypes. In *Proceedings First International Semantic Web Conference (ISWC2002)*, 2002.

[2] DAML-S Coalition: A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. DAML-S: Semantic markup for Web services. In *Proc. International Semantic Web Working Symposium (SWWS)*, pages 411–430, 2001. `http://www.daml.org/services/`.

[3] M. Grüninger. Applications of PSL to semantic web services. In *Proc. of Workshop on Semantic Web and Databases, in association with the Very Large Databases Conference (VLDB)*, 2003.

[4] M. Grüninger and C. Menzel. Process specification language: Theory and applications. *AI Magazine*, 24:63–74, 2003.

[5] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[6] N. Lohmann, H. M. W. Verbeek, C. Ouyang, C. Stahl, and W. M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Tech. Univ. Eindhoven, The Netherlands, August 2007.

[7] S. Narayanan and S. McIlraith. Analysis and simulation of web services. *Computer Networks*, 42:675 – 693, 2003.

[8] R. Reiter. *KNOWLEDGE IN ACTION: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[9] SWSF Committee. Semantic Web Service Ontology (SWSO): First-order Logic Ontology for Web Services (FLOWS), September 9, 2005. `http://www.w3.org/Submission/SWSF-SWSO/`.

[10] SWSF Committee. Semantic Web Services Framework (SWSF) overview, September 9, 2005. `http://www.w3.org/Submission/SWSF/`.

[11] WSMO Committee The Web Service Modeling Ontology (WSMO) Submission, June, 2005. `http://www.w3.org/Submission/2005/06/`.

Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST

Amit P. Sheth, Karthik Gomadam, Ajith Ranabahu
Services Research Lab, kno.e.sis center, Wright State University, Dayton, OH
{amit,karthik, ajith}@knoesis.org

Abstract

Services Research Lab at the Knoesis center and the LSDIS lab at University of Georgia have played a significant role in advancing the state of research in the areas of workflow management, semantic Web services and service oriented computing. Starting with the METEOR workflow management system in the 90's, researchers have addressed key issues in the area of semantic Web services and more recently, in the domain of RESTful services and Web 2.0. In this article, we present a brief discussion on the various contributions of METEOR-S including SAWSDL, publication and discovery of semantic Web services, data mediation, dynamic configuration and adaptation of Web processes. We finally discuss our current and future research in the area of RESTful services.

1 Overview

Our body of research can be divided into three major phases. The first phase related to the METEOR (for "Managing End To End OpeRations") system [6] focused on workflow management and addressed issues of formal modeling, centralized as well as distributed scheduling and execution (including exception handling, security, survivability, scalability and adaptation). The work yielded two notable frameworks:1)WebWork [7], a Web based implementation and 2) ORBWork, a CORBA based implementation. The METEOR project initally started at BellCore in 1990 and was continued at the LSDIS lab until 1998. A commercial spinoff, Infocosm, inc. and the product METEOR EAppS (for Enterprise Application Suite) are other notable accomplishments.

Adopting to the SOA and semantic Web evolution, METEOR evolved into METEOR-S where S stands for services (or Service oriented Architecture) and semantics. It was largely carried out at LSDIS Lab during later 1990s and 2006. One of the significant contributions of METEOR-S research is the submission of WSDL-S specification as a W3C member submission, along with IBM. In 2006, the W3C created a charter for the Semantic Annotation of Web Services (SAWSDL; www.w3.org/ 2002/ws/sawsdl), which used WSDL-S as its primary input. SAWSDL became a W3C candidate recommendation in January 2007.

Our third phase recognizes emergence of Web2.0 and the People Web along with use of microformats for associating metadata to Web resources, and so called light weight web services (RESTful services and WebAPIs). This phase started in 2006 and significantly expanded at the Services Research Lab in Kno.e.sis Center (where our group of 11 researchers moved from the LSDIS lab) in 2007. One of the key initial outcome is a microformat for annotating service descriptions in HTML called hREST and a faceted extension called SA-REST.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Both hREST and SA-REST are in their early stages of research. Further information about these is available at `http://knoesis.wright.edu/research/srl/projects/hRESTs/`.

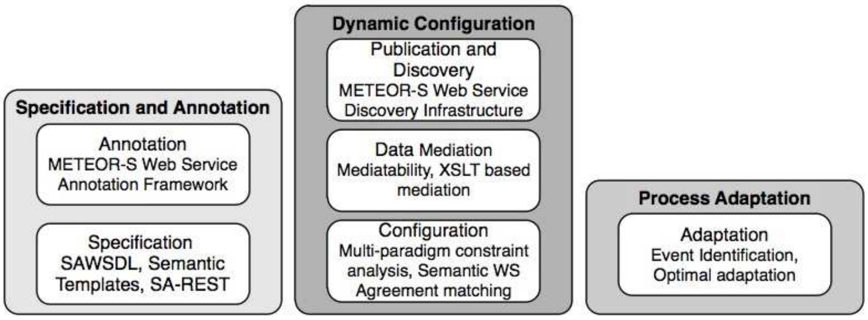


Figure 1: Overview of the various components of the METEOR-S framework.

While other prominent semantic Web service efforts in OWL-S and WSMO have focused on creating service ontologies and process composition to a large extent, the objective of METEOR-S is to define and support the complete life-cycle of Semantic Web processes. METEOR-S adopts an evolutionary approach towards semantic Web services, by extending current SOA (WS-*) standards and specifications to support semantics. We identify three main stages in the life-cycle as illustrated in Figure 1. The first stage comprises of techniques to define annotation mechanisms to extend current SOA standards. Using the annotations to support enhanced discovery and publication of services along with the support for configuration and data-mediation is addressed in the second stage. The third stage addresses identifying events and adapting to various events during execution. We are currently working on exciting area of RESTful services and Web 2.0. More specifically, our research focusses on specifying RESTful services, finding them and integrating them to create smart mashups or smashups. In this article, we present a detailed description of the contributions of the METEOR-S project and briefly describe our current research direction in the area of service oriented computing.

2 Specification and Annotation

The building blocks of SOA-based solutions are self-describing Web services that can be reused across various applications. The Web Service Description Language (WSDL) was created specifically for this purpose and describes the data elements, operations and message bindings. However, WSDL descriptions are not sufficient for the client to unambiguously decipher each operations intended purpose as well as the intended content of its parameters. SAWSDL (which evolved from WSDL-S, first proposed in [13]) overcomes the above limitation by adding semantic meta-data to WSDL elements [15]. Semantic annotations are added to WSDL elements using the *modelreference* extensibility attribute. The *modelreference* value of a WSDL element contains a reference to a concept in the ontology that defines the semantics of that element. This allows service providers to better describe their interfaces and allows clients to better understand the interface descriptions. The original ideas for WSDL-S and SAWSDL were founded on the four types of semantics for services - 1) data semantics: descriptions of the data elements of a service, 2) functional semantics: descriptions of the various operations and functional capabilities of a service, 3) non-functional semantics: descriptions of the non-functional requirements and guarantees, and 4) execution semantics: descriptions of events and faults and how to handle them [10]. We discuss the impact of semantic annotations in realizing dynamic configuration in the next section.

3 Dynamic Configuration

Our research has demonstrated the value of semantic annotations in realizing dynamic SOA environments. The METEOR-S middleware discussed in [5] demonstrates a SOA middleware that supports run time discovery and binding of partner services. Service requirements are both functional and non-functional. Service discovery selects partner services that fulfill the functional requirements. From this set, partners that fulfill the non-functional requirements are selected using constraint analysis.

3.1 Discovery and Publication

Selection of partner services that fulfill the functional requirements of a client is the first step to realize dynamic SOA environments. The METEOR-S Web Service Discovery Infrastructure (MSWDI) is a peer to peer framework for efficiently discovering partner services [12]. MWSDI uses an ontology-based approach to organize registries, enabling semantic classification of all Web services based on domains. Each of these registries supports semantic annotation of the Web services, which is used during discovery process. MWSDI defines four kinds of peers

1. An operator peer controls a Web Service Registry. The role of the Operator peer is to control a registry and to provide Operator services for its registry. The Operator peer also acts as a provider for the Registries Ontology to all other peers who need it.

2. A gateway peer acts as an entry point for registries to join MWSDI. It is responsible for updating the Registries Ontology when new registries join the network. It is also responsible for propagating any updates in the Registries Ontology to all the other peers. Gateway peer is not associated with any registry.

3. Auxiliary peers act as providers of the Registries Ontology.

4. The Client peers are transient members of the peer-to-peer network, as they are instantiated only to allow users to use the capabilities of the MWSDI.

3.2 Multi-Paradigm Constraint Analysis

Partners that fulfill the functional requirements may not fulfill the non-functional requirements. Selecting partners who also fulfill the non-functional requirements is the second step. Non-functional requirements are typically modeled as Service Level Agreements (SLAs). The SLAs lack semantic metadata and are often very generic, thus making it hard to match SLAs from two services. In [9], the authors present a framework to enhance WS-Agreement with structure and semantic metadata. The framework includes a well defined XML based syntax for expressing and semantically annotating SLAs. The additional semantic information allows one to incorporate rules and enables the system to make better matches dynamically.

Non-functional requirements themselves can either be quantitative (*supply time* $\leq$*5 days*) or non-quantitative (*Security must be RSA*). To deal with both, we proposed a multi-paradigm constraint analysis in [1]. The constraint analyzer uses integer linear programming based techniques for optimizing the quantitative constraints and SWRL and SPARQL based techniques for non-quantitative constraints.

3.3 Data Mediation

One of the key benefits of SAWSDL is the systematic approach to data mediation using XSLT. Rather than using XSLT's to mediate between message instances and schemas, SAWSDL advocates mediation at the level of ontologies. To translate a service schema to an ontology, SAWSDL specifies two key techniques - 1) lifting schema mapping and 2) lowering schema mapping. Lifting schema mapping is an XSLT transformation to convert a service schema to an ontology schema. Lowering schema mapping converts an ontology schema into a service schema. To achieve mediation between two service schemas, the source schema is lifted to its corresponding ontology schema. Using schema transformation techniques, the lifted schema is translated into the target ontology schema. This is lowered into the target service schema. The systematic approach offers a huge upgrade in defining and reusing transformation functions [8].

4 Adaptation

The adaptation phase addresses the problem of adapting business processes to runtime events and faults. These include system events such as service unavailability, as well as business level events such as shipment delays. Creating a middleware system with the ability to monitor and adapt to both types of events can be viewed as a two-step problem. The first step is to identify and subscribe to the events to which the system might need to adapt. The second step is to adapt to those events as and when they occur. We present an approach to automatically identify events that may impact the execution of a process in [3] building upon our research in the area of semantic associations for discovering events from a functional and a non-functional ontology.

Once the events are identified, we address adaption as a stochastic decision making problem using Markov Decision Processes(MDP) [14]. MDP policies are generated by using the events identified, event probabilities described by partner services and the cost impact of the events as described in the SLAs. Further, a cost penalty for adaptation is calculated by considering the constraints that occur across services (inter-service dependencies). This allows us to ensure that the adaptation does not violate the process optimality requirements. We discuss three approaches- 1) a centralized approach in which a central controller maintains all the MDP state information, 2) a decentralized approach in which each MDP acts independently of the other in deciding the optimal action and 3) a hybrid approach in which the decentralized MDPs communicate with each other via the central controller.

5 Beyond SOAP: RESTful Services and Web 2.0

Lately, the RESTful services paradigm has gained a lot of traction. Web applications (such as maps and payment processing) and data (such as news feeds) are being exposed as services that can be invoked using scripting languages such as Ruby, PHP and Javascript. Web application hybrids or mashups have emerged as a very popular way for integrating RESTful services. Despite their popularity, the programming complexity and the fundamental problem of data mediation make it hard for non-expert developers to create meaningful mashups. Our research in the area of RESTful services addresses this limitation. We break down our approach into three steps: 1) specification, 2) finding the right set of services and 3) Service integration.

In the area of specification, we are advocating a new microformat called hRESTs for service descriptions in HTML. hRESTs provides constructs to markup operations and data elements in an API description. hRESTs evolved from our current work on SA-REST, first proposed in [11] and inherited the operation and data element constructs of SA-REST. Furthermore, in addition to operations and data elements, RESTful API descriptions have other facets such as data formats (JSON, GData), and client library bindings (Java, PHP). These are captured using the constructs of SA-REST, which is being modeled as an extension to hRESTs. A more detailed description of hRESTs can be found at: http://knoesis.wright.edu/research/srl/projects/hRESTs/.

RESTful services are often described as Web APIs using HTML. The lack of a model like WSDL makes it difficult to use conventional service discovery approaches. Currently general purpose search engines such as Google are often used for finding these APIs. API search frameworks such as programmableWeb rely on user classification and often yield poor results. In our research, we use traditional text classification techniques for faceted classification and indexing of APIs. We also have developed a ranking algorithm similar to PageRank called Service Utilization (ServiUt rank) for ranking APIs [2]. Finally, in the area of integration, we currently focus on the problem of data mediation. Though there have been numerous attempts to realize automatic mediation, there is still considerable amount of human effort required in the process. We define a metric called Mediatability that estimates the amount of human effort needed in mediation. The mediatability computation algorithm is a two pass algorithm that uses the concept of nearest common parent, first proposed by Tarjan. The first pass is a top down pass that computes the matching values and the similarity of the two schema trees. The second pass is a bottom up pass that computes the mediatability values using the matching and similarity values [4].

6 Conclusions

In addition to proposing newer techniques and standards, the METEOR-S research has also contributed open source software for handling SAWSDL object models (SAWSDL4J, Woden4SAWSDL), semantic annotation (Radiant) and for discovery and publication (Lumina). Much of the past work in the area of semantic Web services has focused on the WS-* implementation of SOA. Lately, the RESTful approach to SOA has gained popularity, largely due to its lightweight approach. We are currently working on the specification, search and integration of RESTful services and Web APIs. It is our belief that our current research would ease the task of creating mashups and would allows users to create customizable and dynamically configurable smart mashups.

Acknowledgements: We acknowledge the contributions of Professor John Miller, Dr. Kunal Verma and other members of the METEOR-S project at LSDIS lab.

References

[1] R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *IEEE SCC*, pages 23–30, 2004.

[2] K. Gomadam, A. Ranabahu, M. Nagarajan, K. Verma, and A. P. Sheth. A faceted classification based approach to search and rank web apis. In *ICWS*, page To Appear., 2008.

[3] K. Gomadam, A. Ranabahu, L. Ramaswamy, A. P. Sheth, and K. Verma. A semantic framework for identifying events in a service oriented architecture. In *ICWS*, pages 545–552, 2007.

[4] K. Gomadam, A. Ranabahu, L. Ramaswamy, K. Verma, and A. P. Sheth. Mediatability: Estimating the degree of human involvement in xml schema mediation. In *ICSC*, page To Appear., 2008.

[5] K. Gomadam, K. Verma, A. P. Sheth, and J. A. Miller. Demonstrating dynamic configuration and execution of web processes. In *ICSOC*, pages 502–507, 2005.

[6] N. Krishnakumar and A. P. Sheth. Managing hetergeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):155–186, 1995.

[7] J. A. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh. Webwork: Meteor's web-based workflow management system. *Journal of Intelligent Information Systems*, 10(2):186–215, 1998.

[8] M. Nagarajan, K. Verma, A. P. Sheth, J. A. Miller, and J. Lathem. Semantic interoperability of web services - challenges and experiences. In *ICWS*, pages 373–382, 2006.

[9] N. Oldham, K. Verma, A. P. Sheth, and F. Hakimpour. Semantic ws-agreement partner selection. In *WWW*, pages 697–706, 2006.

[10] A. Sheth. Semantic web process lifecycle: Role of semantics in annotation, discovery, composition and orchestration, 2003.

[11] A. P. Sheth, K. Gomadam, and J. Lathem. Sa-rest: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.

[12] K. Sivashanmugam, K. Verma, and A. P. Sheth. Discovery of web services in a federated registry environment. In *ICWS*, pages 270–278, 2004.

[13] K. Sivashanmugam, K. Verma, A. P. Sheth, and J. A. Miller. Adding semantics to web services standards. In *ICWS*, pages 395–401, 2003.

[14] K. Verma, P. Doshi, K. Gomadam, J. A. Miller, and A. P. Sheth. Optimal adaptation in web processes with coordination constraints. In *ICWS*, pages 257–264, 2006.

[15] K. Verma and A. P. Sheth. Semantically annotating a web service. *IEEE Internet Computing*, 11(2):83–85, 2007.

Process Mediation, Execution Monitoring and Recovery for Semantic Web Services

Katia Sycara
The Robotics Institute
Carnegie Mellon University
katia@cs.cmu.edu

Roman Vaculín
Institute of Computer Science
Academy of Sciences of the Czech Republic
vaculin@cs.cas.cz

1 Introduction

One of the main promises of web services standards is to enable and facilitate seamless interoperability of diverse applications and business processes implemented as components or services. A service can be part of a business workflow that prescribes control and data flows of complex applications. As business needs change, processes may need to get reconfigured or additional process components and services may need to be added. As a result of these changes, the previous components must become interoperable with the new one. This can be accomplished by making manual changes to the existing workflow components and programming the new components in such a way as to have interoperability built-in. This is a rather laborious and inefficient process since it must be repeated every time workflow reconfiguration is needed. Also, since many different elements of a business workflow may be under the control of third parties (e.g. subcontractors), additional costly coordination will be needed with these third parties to manually find interoperability solutions. Moreover, since the Internet gives the opportunity to dynamically discover service providers, it is often not a priori known which service provider may best fit the application workflow changing needs. In other words, the new service component that must interoperate with old ones, is dynamically discovered. Therefore, (a) a more general solution is desired, namely the ability to achieve process interoperability (e.g. interoperability of existing processes with new ones) without actually modifying their implementation and interfaces, and (b) the mediation may need to be done dynamically even at runtime, which implies that only minimal assumptions about knowledge of service requester and service provider interfaces is allowed. One solution to this requirement is to apply a process mediation component which resolves all incompatibilities and generates appropriate mappings between different processes while making minimal assumptions about implementation details of service providers and requesters.

Creating such a process mediation component is a very challenging task. Service providers and requesters may not share basic standards for Web Service specification; they may not share domain ontologies; furthermore, they typically do not do share the same data models or interaction protocols. Moreover, the changing business needs may dictate that existing services are modified, thus rendering previous compatible interactions incompatible. As a result, a mediation module must deal with incompatibilities of multiple types and also be able to incorporate adaptive reasoning mechanisms to address dynamic environment changes.

Current web services standards provide a good basis for achieving at least some level of interoperability. WSDL allows to declaratively describe operations and format of messages and data structures that are used to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

communicate with the web service. BPEL4WS adds the possibility to define the interaction protocol and possible control flows and combine several web services within a formally defined process model. However, none of the current standards goes beyond the syntactic descriptions of web services. Newly emerging standards for semantic web services, such as SAWSDL [4], OWL-S [8] and WSMO [6], strive to enrich syntactic specifications with rich semantic annotations to further facilitate flexible dynamic web services discovery, composition and invocation [7]. However, the current standards do not provide reasoning methods for interoperability of providers and requesters as application requirements change. Various types of middle agents [14] – employing techniques such as reasoning and planning combined with approaches like dynamic discovery and recovery from failure – present a possible solution for bridging the gap between service requesters and providers with incompatible interaction protocols (process models) and possibly incompatible data models.

2 Process Mediation

In our recent body of work [11, 9], we address the problem of automatic mediation of process models consisting of semantically annotated web services. Processes can act as service providers, service requesters or communicate in peer-to-peer fashion. We are focusing on the situation where the interoperability of two components, one acting as the requester and the other as the provider, needs to be achieved. Usually, both the requester and provider adhere to some relatively fixed process models. The process models can either correspond to a particular existing implementation or they can be default (generic) process models that for example generalize a business processes of some specific problem domain (e.g., client or provider of flight booking service). In particular, our research focuses on mediation of process models of providers and requesters in open dynamic environments where new services could be dynamically discovered, thus necessitating runtime mediation. Additionally, we assume that both the requester and the provider interact according to specified process models that are fixed, are expressed declaratively and might be incompatible.

We use the OWL-S ontology [8] for semantic annotations because it provides support for description of individual services and also explicit constructs with clear semantics for describing process models. In OWL-S, the elementary unit of process models is an atomic process, which represents one indivisible operation that the client can perform by sending a particular message to the service and receiving a corresponding response. Processes are specified by means of their inputs, outputs, preconditions, and effects (IOPEs). Types of inputs and outputs are defined as concepts in an ontology or as simple XSD data-types. Processes can be combined into composite processes by using control constructs such as sequence, any-order, if-then-else, split, loops, etc.

Creating a mediator component is very challenging since this component must resolve various types of mismatches, such as the following that we have identified:

A. Data level mismatches: e.g. data are represented as different lexical elements (numbers, dates format, local specifics, etc.); or ontological mismatches

B. Service level mismatches: e.g. a requester's service call is realized by several provider's services or a sequence of requester's calls is realized by one provider's call; some information required by the provider is not provided by the requester; information provided by one party is not needed by the other one

C. Protocol / structural level mismatches: e.g. control flow in the requester's process model can be realized in different ways in the provider's model (e.g., sequence can be realized as an unordered list of steps, etc.)

We have developed an abstract process mediation framework (APFM) showed in Figure 1. The main goal of the APMF is a clear identification and separation of critical functional areas which need to be addressed by mediation components in order to effectively solve the process mediation problem. The three key functionalities, namely *process mediation*, *data mediation* and *service invocation*, are displayed as horizontal layers. The process mediation layer, realized by process mediators, is responsible for resolving service level and protocol level mismatches (categories B and C). The data mediation layer, realized by data mediators, is responsible for resolving data level mismatches (category A). Typically, when trying to achieve interoperability, process mediators and data mediators are closely related. A natural way is to use data mediators within the process mediation component to resolve "lower" level mismatches that were identified during the process mediation. The service invocation layer is responsible for interactions with actual web services, which include the services of the requester, provider and possibly other external services.

To address runtime incompatibilities and possible service failures, the mediation processes make use of *monitoring* and *recovery* functionalities, which are represented as vertical layers in Figure 1. Finally, in dynamic environments *discovery of external services* is closely related to the process mediation since external services (e.g. a translation service between inches and meters) might need to be discovered which are capable of delivering information for resolving mismatches identified between two processes.

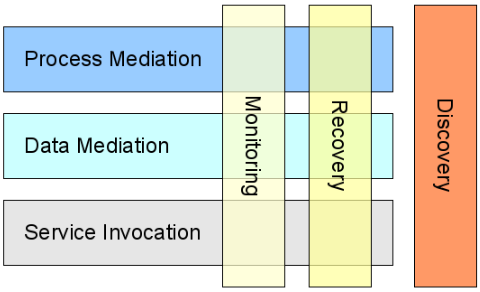


Figure 1: Abstract Process Mediation Framework

We have investigated and developed concrete architectures for the mediation components in the APMF framework for two cases: (a) when the mediation component has *complete visibility* of the process model of the service provider and the service requester [11] or (b) when the mediation component has *visibility only of the provider's process model* but not the requester's (we call this asymmetric visibility) [9].

In the complete visibility scenario, our solution is based on an off-line analysis of possible execution sequences of the requester. A planning algorithm is employed to identify mismatches between requester's execution sequences and the provider's process model, and to compute the appropriate mappings for bridging the identified mismatches. Such mappings are used during +runtime mediation to perform the necessary translations. In the case of asymmetric visibility, the off-line analysis cannot be employed because the requester's process model is not available. Therefore, the mediation must rely strictly on computing the mapping during runtime only. We have developed a process mediation agent that uses similar planning techniques as in the complete visibility scenario except that the planning is constrained by time due to the requirement of a timely response. Additionally, the process mediation agent incorporates advanced recovery techniques to deal both with service failures and with possible wrong choices made during the mediation.

3 Semantic Monitoring

We have developed an ontology [12] for specification of primitive events and a language for specification of composite event patterns [10] based on the event algebra developed originally in the context of active databases [3, 2]. Additionally, we have developed monitoring mechanisms combined with introspection mechanisms and error handling that we implemented as extensions of the OWL-S Virtual Machine [5] which is a component that controls interactions between the clients and semantic web services. Specifically, the OWL-S Virtual Machine (OVM) executes the process model of a given service by going through the process model while respecting the

OWL-S operational semantics [1] and invoking individual services represented by atomic processes. During the execution, the OVM processes inputs provided by the requester and outputs returned by the provider's services, realizes the control and data flow of the composite process model, and uses the grounding to invoke WSDL based web services when needed. The OVM is a generic execution engine which can be used to develop applications that need to interact with OWL-S web services. During the service execution, the execution engine (OVM) emits events specific to the state of process model execution. Emitted events (primitive events) are instances of generic event or fault types defined in the events ontology [12]. The content of emitted event instances describes the execution context in the time when the event occurred and other information relevant to the given event type. The content is semantically annotated by the same domain ontology concepts that are used in the service definition itself, which allows a more flexible events detection techniques than those derived from a simple syntactic keywords matching. Specifically, we employ semantic reasoning for detecting primitive events based on matching their event type and the content.

The implemented monitoring extensions allow to perform different monitoring tasks such as logging, performance measuring, execution progress tracking, execution debugging or evaluations of security parameters. For many applications simple detection of individual events (called primitive events) emitted by various components of the systems is a sufficient solution. However, often complex events patterns (called composite events) such as co-occurrence of different events or sequence of events need to be detected.

4 Fault Handling and Error Recovery

Currently, neither WSMO, nor OWL-S provide any support for fault handling and recovery. The ability to handle failures correctly and to possibly be able to recover from failures is important not only in the context of process mediation, but for web services in general. We have developed techniques for fault handling and recovery for semantic web services [13] to allow specification of reliable, possibly adaptive process models and so to increase the autonomy of web services systems. Again, we focus primarily on dynamic environments where cooperating services might need to be discovered during runtime. Our approach to fault handling and recovery shares similarities with fault handling in WS-BPEL. However, WS-BPEL offers only a limited support for recovery and the monitoring which makes it suitable rather for static scenarios.

The basic idea of our approach is to take advantage of powerful semantic monitoring techniques to define and detect possible erroneous states. To allow a controlled process recovery and gradual execution degradation standard *fault handling* must be augmented with mechanisms allowing a *designer* to define what situations are supposed to trigger an erroneous state. To achieve this, we augment the process model definition with constraint violation handlers (*CV-handlers*) for associating constraint violation conditions with appropriate explicit *recovery actions* that resolve the violations. Such constraints can stem from applicable SLAs or from contractual requirements. Constraint violation conditions are treated as hard constraints that lead to an abnormal execution state. To express soft constraints that do not necessarily lead to an erroneous state, we use *event handlers*. A condition part of both event handlers and CV-handlers must be expressive and intuitive enough to allow encoding of SLAs and other constraints. We have employed event algebra expressions [2] combined with semantic filters [10], which are suitable for describing complex event patterns and allow an efficient events monitoring and detection (described briefly in the previous section). Similarly to WS-BPEL, we use *compensation* for undoing effects of the partial work after a fault has occurred. Finally, we introduced *explicit recovery actions* (such as *retry*, *replaceBy*, *replaceByEquivalent*) as means of fixing problems manifested by the fault occurrence. Recovery actions present means of restoring the normal execution flow.

5 Acknowledgments

This research was supported in part by Darpa contract FA865006C7606 and in part by funding from France Telecom. Research partially supported by the Czech Science Foundation project 201/05/H014 and the Czech Ministry of Education project ME08095.

References

[1] Anupriya Ankolekar, Frank Huch, and Katia P. Sycara. Concurrent semantics for the web services specification language DAML-S. In Farhad Arbab and Carolyn L. Talcott, editors, *COORDINATION*, volume 2315 of *Lecture Notes in Computer Science*, pages 14–21. Springer, 2002.

[2] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In Giorgio C. Buttazzo, editor, *EMSOFT*, pages 147–154. ACM, 2004.

[3] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 606–617, Santiago, Chile, 1994.

[4] Joel Farrell and Holger Lausen. Semantic annotations for WSDL and XML schema, 2007. `http://www.w3.org/TR/sawsdl/`.

[5] Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia P. Sycara. The DAML-S virtual machine. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2003.

[6] Dumitru Roman et al. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77 – 106, 2005.

[7] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1 (1):27–46, 2004.

[8] The OWL Services Coalition. *Semantic Markup for Web Services (OWL-S)*. `http://www.daml.org/services/owl-s/1.1/`.

[9] Roman Vaculín, Roman Neruda, and Katia Sycara. An agent for asymmetric process mediation in open environments. In *Service Oriented Computing: Agents, Semantics and Engineering*, pages 1032–1039. Springer Verlag, May 12-16 2008.

[10] Roman Vaculín and Katia Sycara. Specifying and monitoring composite events for semantic web services. In *The 5th IEEE European Conference on Web Services*. IEEE Computer Society, November 26-28 2007.

[11] Roman Vaculín and Katia Sycara. Towards automatic mediation of OWL-S process models. In *2007 IEEE International Conference on Web Services*, pages 1032–1039. IEEE Computer Society, July 9-13 2007.

[12] Roman Vaculín and Katia Sycara. Semantic web services monitoring: An OWL-S based approach. In *41st Hawaii International Conference on System Sciences*. IEEE Computer Society Press, January 7-10 2008.

[13] Roman Vaculín, Kevin Wiesner, and Katia Sycara. Exception handling and recovery of semantic web services. In *Fourth International Conference on Networking and Services*. IEEE Computer Society Press, 2008.

[14] H. Chi Wong and Katia P. Sycara. A taxonomy of middle-agents for the internet. In *ICMAS*, pages 465–466. IEEE Computer Society, 2000.

Automatic Service Composition and Synthesis: the Roman Model

Diego Calvanese
Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
calvanese@inf.unibz.it

Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, Fabio Patrizi
Dipartimento di Informatica e Sistemistica, SAPIENZA – Università di Roma, Italy
lastname@dis.uniroma1.it

Abstract

The promise of Web Service Computing is to use Web services as fundamental elements for realizing distributed applications/solutions. When no available service satisfies a desired specification, one might check whether (parts of) available services can be composed and orchestrated in order to realize the specification. The problem of automatic composition becomes especially interesting in the presence of conversational services. Among the various frameworks proposed in the literature, here we concentrate on the so called "Roman Model", where: (i) each service is formally specified as a transition system that captures its possible conversations with a generic client; (ii) the desired specification is a target service, described itself as a transition system; (iii) the aim is to synthesize an orchestrator realizing the target service by exploiting execution fragments of available services. The Roman Model well exemplifies what can be achieved by composing conversational services and, also, uncovers relationships with automated synthesis of reactive processes in Verification and AI Planning.

1 Introduction

Web services, or simply services, are modular applications that can be described, published, located, invoked, and composed over a variety of networks (including the Internet): any piece of code and any application component deployed on a system can be wrapped and transformed into a network-available service, by using standard (XML-based) languages and protocols (e.g., WSDL, SOAP, etc.). One of the interesting aspects is that this wrapping allows each program to export a simplified description of itself, which abstracts from irrelevant programming details. The promise of Web services is to enable the composition of new distributed applications/solutions: when no available service can satisfy a client request, (parts of) available services can be composed and orchestrated in order to satisfy the request itself.

The work on services has by now largely resolved the basic interoperability problems for service composition (e.g., standards such as WS-BPEL and WS-CDL exist and are widely supported in order to compose services), and designing programs, called orchestrators, that execute compositions by coordinating available services according to their exported description is the bread and butter of the service programmer [1].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

The availability of abstract descriptions of services, has been instrumental to devising automatic techniques for synthesizing service compositions and orchestrators. Several research lines have been opened to investigate this issue. Some works have concentrated on data-oriented services, by binding service composition to the work on data integration [21]. Other works have looked at process-oriented services, in which operations executed by the service have explicit effects on the system. Among these approaches, several consider *stateless* (a.k.a., atomic) services, in which the operations that can be invoked by the client do not depend on the history of interactions, as services do not retain any information about the state of such interactions. Much of this work relies on the literature on Planning in AI [30, 10, 12]. Others consider *stateful* services which impose some constraints on the possible sequences of operations (a.k.a., conversations) that a client can engage with the service. Composing stateful services poses additional challenges, as the composite service should be correct w.r.t. the possible conversations allowed by the component ones. Moreover, when dealing with composition, data (that typically are sent back and forward in the operation invocations and are manipulated by the service) usually play an important role. This work relies on research carried out in different areas, including research on Reasoning about Actions and Planning in AI, and research about Verification and Synthesis in Computer Science [11, 25, 18, 20].

In this paper, we focus on composition of process-oriented stateful services, in particular we consider the framework for service composition adopted in [5, 7, 8, 22, 16, 29], sometimes referred to as the "Roman Model" [19]. In the Roman Model, services are represented as transition systems (i.e., focusing on their dynamic behavior) and the composition aims at obtaining, given a (virtual) target service specifying a desired interaction with the client, an actual composite service that preserves such an interaction.

The Roman Model well exemplifies what can be achieved by composing stateful services, and allows to uncover relationships with automated synthesis of reactive processes in Verification and Planning in AI.

2 The Roman Model

Services in the Roman Model represent software modules capable of performing operations. They are *stateful*: a service, at each step, offers to its clients a choice of operations it can perform, based upon its own state; the client chooses one of the offered operations, and the service executes it, changing its state accordingly. Formally, a *service* is a transition system $\mathcal{S} = \langle \mathcal{O}, S, s^0, S^f, \varrho \rangle$, where: (*i*) $\mathcal{O}$ is the set of possible *operations* that the service recognizes; (*ii*) S is the finite set of service's *states*; (*iii*) $s^0 \in S$ is the *initial state*; (*iv*) $S^f \subseteq S$ is the set of *final states*, i.e., those states where the interaction with the service can be legally terminated by the client (though she does not need to); (*v*) $\varrho \subseteq S \times \mathcal{O} \times S$ is the service's *transition relation*, which accounts for its state changes. When $\langle s, o, s' \rangle \in \varrho$, we say that *transition* $s \xrightarrow{o} s'$ *is in* $\mathcal{S}$. Given a state $s \in S$, if there exists a transition $s \xrightarrow{o} s'$ in $\mathcal{S}$, then operation o is said to be *executable* in s. A transition $s \xrightarrow{o} s'$ in $\mathcal{S}$ denotes that s' is a possible successor state of s, when operation o is executed in s. Notice that we allow for *nondeterministic* services, that is, several transitions can take place when executing a given operation in a given state. So, when choosing the operation to execute next, the client of the service cannot be certain of which choices will be available later on, this depending on which transition actually takes place. In other words, nondeterministic services are only *partially controllable*. We say that a service $\mathcal{S}$ is *deterministic* iff there are no two distinct transitions $s \xrightarrow{o} s'$ and $s \xrightarrow{o} s''$ such that $s' \neq s''$. Notice that given a deterministic service's state and an executable operation in that state, *unique* next service's state is always known. That is, deterministic services are indeed *fully controllable* by just selecting the operation to perform next.

A *community of available services* $\mathcal{C} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_n \rangle$ consists of n nondeterministic available services that share the same operations $\mathcal{O}$. A *target service* is a desired *deterministic* service that shares the operations in $\mathcal{O}$. The requirement of being deterministic is due to the fact that we want such a service to be fully controllable by its clients. The goal of the composition in the Roman Model is to maintain with the client the same, possibly infinite, interaction that she would have with the (virtual) target service, by suitably orchestrating the (concrete) available services. An *orchestrator* is a system component able to activate, stop, and resume any of the available services, and to instruct them to execute an operation among those executable in their current state. Essentially, the orchestrator, at each step, will consider the operation chosen by the client (according to the target service) and delegate it to one of the services for which the operation is executable, on so on, possibly at infinitum. The aim of the orchestrator is to maintain the interaction with the client, as if it was interacting with the target service, without ever failing to be able to delegate an operation chosen by the client to one of the available services. We assume here that the orchestrator has *full observability* on the available services, that is, it can keep track (at runtime) of their current states. Although other choices are possible, full observability is the natural one in this context, since the available services, modeled through finite transition systems as above, are already suitable abstractions for *actual* modules: if details have to be hidden, this can be done directly within the abstract behaviors exposed by services, possibly exploiting nondeterminism.

Formally, an orchestrator is a *function* from (*i*) the *history* of the whole system (which includes the state trajectories of all available services and the trace of the operations chosen by the client, and executed by the services), and (*ii*) the *operation* currently chosen by the client, to the index i of the service $\mathcal{S}_i$ to which the operation has to be delegated. Intuitively, the orchestrator *realizes* a target service if and only if, at every step, given the current history of the system, it is able to delegate every operation executable by the target to one of the available services.

3 Composition techniques

The goal of service composition is to synthesize an orchestrator that realizes the target service by exploiting available services. Such problem is related to synthesis of reactive processes [27], where an environment (in our case, the available service community) is to be controlled by an automatically-generated controller (in our case, the orchestrator), so that a desired specification (in our case, mimicking the target service) is fulfilled.

The specific composition problem has been tackled with different techniques: at first by exploiting a reduction to Satisfiability in a well known logic of programs, namely PDL [5, 7, 4, 16]. Notably, Logics of Programs are tightly related to Description Logics, for which highly optimized satisfiability checkers exist (e.g., RacerPro, Pellet, FACT, etc.). More recently [23], the problem has been tackled by directly appealing to techniques for Linear Time Logic (LTL) synthesis [26], based on model checking of game structures for the so called *safety-games* (see also ATL [3, 2]). Another approach recently proposed is based on directly computing compositions by exploiting (variants of) the formal notion of simulation [9, 29, 23]. The two latter approaches promise both a high level of scalability, since in practice they can be based on symbolic model checking technologies. Here we concentrate on the simulation-based approach.

Let $\mathcal{C} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_n \rangle$ be a community of available services and $\mathcal{S}_t$ a target service, where $\mathcal{S}_i = \langle S_i, s_i^0, S_i^f, \varrho_i \rangle$, for $i \in \{t, 1 \ldots, n\}$. An *ND-simulation relation* of $\mathcal{S}_t$ by $\mathcal{C}$ is a relation $R \subseteq S_t \times S_1 \times \ldots \times S_n$ such that $\langle s_t, s_1, \ldots, s_n \rangle \in R$ implies that if $s_t \in S_t^f$ then $s_i \in S_i^f$, for $i \in \{1, \ldots, n\}$, and for each $o \in \mathcal{O}$, there exists a $k \in \{1, \ldots, n\}$ such that for all transitions $s_t \xrightarrow{o} s_t'$ in $\mathcal{S}_t$ we have that: (*i*) there exists a transition $s_k \xrightarrow{o} s_k'$ in $\mathcal{S}_k$; (*ii*) for all $s_k \xrightarrow{o} s_k'$ in $\mathcal{S}_k$, it holds that $\langle s_t', s_1, \ldots, s_k', \ldots, s_n \rangle \in R$. An ND-simulation is essentially a simulation between $\mathcal{S}_t$ and the asynchronous product of the services $\mathcal{S}_i$ in $\mathcal{C}$. However, differently from the usual notion of simulation, we need to take into account available services' nondeterminism. To this end, we require that (*i*) for each target service's operation an available service k can be selected to perform the operation and (*ii*) *all its successor states* are still included in the ND-simulation.

A state s_t *is ND-simulated by* $\langle s_1, \ldots, s_n \rangle$, denoted $s_t \preceq \langle s_1, \ldots, s_n \rangle$, if and only if there exists an ND-simulation R of $\mathcal{S}_t$ by $\mathcal{C}$ such that $\langle s_t, s_1, \ldots, s_n \rangle \in R$. Observe that this is a *coinductive definition*. As a result, the relation $\preceq$ is itself an ND-simulation, and is in fact the *largest ND-simulation relation*, i.e., all ND-simulation relations are contained in $\preceq$. It can be shown that there exists a compositions if and only if $s_t^0 \preceq \langle s_1^0, \ldots, s_n^0 \rangle$.

Synthesizing composition using simulation has a very interesting property: the maximal simulation $\preceq$ contains enough information to allow for extracting every possible composition, through a suitable choice function. This allows for devising compositions in a "just-in-time" fashion: we compute the maximal simulation then, based on it, we start executing the composition, choosing the next step according to criteria that can depend on information available at run-time (actual availability of services, network communication problems or cost, etc.), so that simulation is preserved. This, also, opens up the possibility of having failure resistant compositions that reactively or parsimoniously adjust to failures of available services, avoiding recomputing the whole composition from scratch [29].

4 Conclusion

Several extensions and variants of the model presented here have been studied, e.g.: forms of target service's loose specifications [6], lookahead [14], trust aware services [13], distributed orchestrators [28], shared environments or other infrastructure for communication among services [16, 15], data-aware services [4]. Also, the approach described in this paper is related to composition based on planning [25], where the crucial difference is the desired specification to realize: in the composition via planning, this is a desired state of affair to be reached after some interactions while, in our case, it amounts to indefinitely maintain the specified interaction itself.

We conclude by stressing out that *dealing with data* is certainly one of the most critical and difficult issues we currently face in service composition and, more generally, in process verification. Indeed, current verification and synthesis techniques apply to finite state systems, while the presence of data typically results in infinite states. Therefore, suitable means for *abstraction* from infinite to finite states are needed, and indeed virtually all results on combining data with processes are directly or indirectly based on such a notion [4, 17, 24].

Acknowledgements. This work has been partly supported by the IST projects TONES, SemanticGOV, SM4All, the Italian FIRB project TOCAI.it and the IBM 2008 Faculty Award.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[3] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. of CAV 1998*.

[4] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. of VLDB 2005*.

[5] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-Services that export their behavior. In *Proc. of ICSOC 2003*.

[6] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of underspecified composite e-Services based on automated reasoning. In *Proc. of ICSOC 2004*.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *International Journal of Cooperative Information Systems*, 14(4), 2005.

[8] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Composition of services with nondeterministic observable behavior. In *Proc. of ICSOC 2005*.

[9] D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.

[10] J. Blythe and J. L. Ambite, editors. *Proc. of ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[11] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proc. of WWW 2003*.

[12] J. Cardoso and A. Sheth. Introduction to semantic web services and web process composition. In *Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*.

[13] F. Cheikh, G. De Giacomo, and M. Mecella. Automatic web services composition in trustaware communities. In *Proc. of SWS 2006*.

[14] Z. Dang, O. H. Ibarra, and J. Su. On composition and lookahead delegation of e-services modeled by automata. *Theor. Comput. Sci.*, 341(1–3):344–363, 2005.

[15] G. De Giacomo, M. de Leoni, M. Mecella, and F. Patrizi. Automatic workflows composition of mobile services. In *Proc. of ICWS 2007*.

[16] G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI 2007*.

[17] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.

[18] C. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated composition of e-services: Lookaheads. In *Proc. of ICSOC 2004*.

[19] R. Hull. Web services composition: A story of models, automata, and logics. In *Proc. of ICWS 2005*.

[20] S. A. McIlraith and T. C. Son. Adapting Golog for composition of semantic web services. In *Proc. of KR 2002*.

[21] M. Michalowski, J. L. Ambite, C. A. Knoblock, S. Minton, S. Thakkar, and R. Tuchinda. Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

[22] A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proc. of FoSSaCS 2007*, 2007.

[23] F. Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA – Università di Roma, Dipartimento di Informatica e Sistemistica, 2008. To appear.

[24] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *Proc. of IJCAI 2005*.

[25] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proc. of ICAPS 2005*.

[26] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive designs. In *Proc. of VMCAI 2006*.

[27] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL'89*.

[28] S. Sardina, F. Patrizi, and G. De Giacomo. Automaticsynthesis of a global behavior from multiple distributed behaviors. In *Proc. of AAAI 2007*.

[29] S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc. of KR 2008*.

[30] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proc. of ISWC 2003*.

Automated Composition of Web Services: the ASTRO Approach

Annapaola Marconi, Marco Pistore, and Paolo Traverso
FBK-irst - via Sommarive 18, 38100, Trento, Italy
[marconi,pistore,traverso]@fbk.eu

With automated composition we mean generating an executable process that satisfies a given composition requirements by communicating with a set of existing Web services. Several approaches have been proposed to tackle this problem. However, most of them either omit or oversimplify important aspects of the Web service composition problem. The driving idea of the approach we're presenting in this paper is to overcome these limitations, in order to deal with real world composition problems. The ASTRO approach is able to cope with complex control and data flows, i.e., with Web services exposing complex protocols and exchanging structured data, and with composition requirements expressing constraints not only on the service interactions but also on the exchanged data. The ASTRO approach has been implemented and evaluated on real world composition domains.

1 Introduction

The ability to compose services, reducing development time and effort by re-using existing functionalities, is one of the most promising ideas underlying Web services. However, the complexity of service-based applications, the heterogeneity of the components, the dynamic nature of the environment, and the intrinsic distributed structure of the systems, make the manual development of the new composite application a difficult, error-prone and time-consuming task. Given this, techniques and methods allowing to automatically compose and adapt Web services are essential to substantially decrease time and costs in the development, integration, and maintenance of complex service oriented applications.

With automated composition we mean generating an executable process that satisfies a given composition requirements by communicating with a set of existing Web services, and that can be published itself as a Web service providing new higher level functionalities. Several approaches have been proposed to tackle this problem (e.g. [4, 2, 3, 12, 5]). However, most of them either omit or oversimplify several important aspects of the Web service composition problem. The main aim of the ASTRO approach is to overcome these limitations by providing an automated composition framework that is able to tackle real world Web service composition problems.

Among the most important characteristics provided by this approach are (i) the ability to consider component services that are complex stateful processes exhibiting an asynchronous and non deterministic behavior, (ii) the possibility to specify composition requirements specifying both data and control constraints on the execution of the new composite service, and (iii) the possibility to gradually refine the composition requirements and to iteratively re-generate a solution in a continuous semi-automated composition process.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

The ASTRO approach is implemented and incorporated into a prototype tool that supports all the phases of Web service automated composition: from the specification of control-flow and data-flow requirements by means of graphical tools for drawing data net diagrams and specifying control-flow requirements, to the automatic synthesis of the desired service, to the deployment, simulation, and execution of the new composite service. Using the prototype implementation, we evaluated our framework on a set of real-world case studies emerged from industrial applications and found in the literature. A significant example is the combination of Amazon on-line shopping services with on-line payment services provided by banks.

2 An Overview of the Approach

The ASTRO approach conceives the automated synthesis of the composite process as a step of a more complex iterative process that covers the different phases of the composition problem. In particular, the ASTRO composition process [8] consists of two phases (see Figure 1). The aim of the **first phase** is to obtain a preliminary version of the composite process starting from initial composition requirements. During this phase the developer analyzes the component service protocols (abstract WS-BPEL and WSDL) and specifies control flow and data flow requirements. Given the description of the component services and the requirements specification we automatically generate the internal executable composite process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL). This preliminary version of the composite service can be iteratively enhanced in the **second phase** of the process. During this phase the developer, on the basis of the automated composition outcomes, can refine both the composition requirements and the customer interface and automatically re-compose.

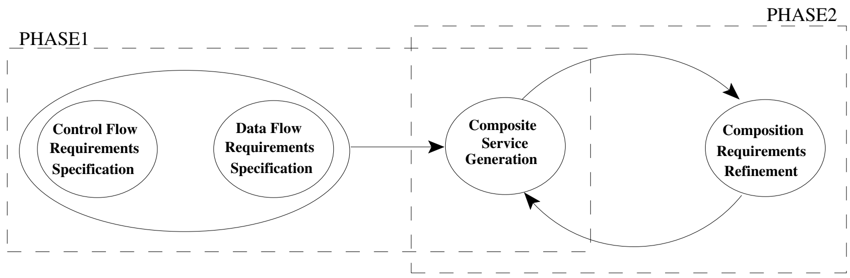


Figure 1: ASTRO Web Service Composition Process

2.1 Specification of Composition Requirements

In order to cope with a wide range of composition problems we need a way to express requirements that define complex conditions, both for what concerns the behavior of the composition (termination conditions, failure recovery, transactional constraints) and for the data exchanged among the component services. Moreover, to make the automated composition an effective and practical task, the requirements specification should be as user-friendly and easy refining as possible.

We propose to separate the specification of data flow requirements from that of control flow requirements, and provide formal notations for their specification. In particular, in the control flow requirement specification step the developer defines termination conditions and transactional issues by exploiting minimal semantic annotations in the component service abstract WS-BPEL. Our approach, described in details in [9, 10], provides the developer with the ability to specify with a simple tabular notation these requirements that are then automatically translated into a formal internal notation that allows for the automation of the composition task. For what concerns requirements on data, we propose a formal language, the data net language [6], that allows to specify complex data flow composition requirements through an intuitive graphical notation. The data flow requirement specification step concerns the specification of how incoming messages must be used by the composite service (from simple forwarding to complex data manipulation) to obtain outgoing messages. During this step the developer also specifies messages received from and sent to the composite service user.

2.2 Automated Composition

Given the description of the component services and the composition requirements, the final step of the first phase is the development of the new composite service. The outcome of this completely automated phase is the executable WS-BPEL implementing the internal behavior of the new process and the description of the interaction protocol that the new service expects its customers to follow (WSDL and abstract WS-BPEL). For this automated synthesis task, the approach exploits sophisticated AI techniques for planning in asynchronous domains, extending them with new methods and algorithms in order to handle the peculiarities of the Web service automated composition problem. In particular, component services define the planning domain, composition requirements are formalized as planning goal, and planning algorithms are used to generate the composite service.

The formal framework, presented in [9], differs from other planning frameworks since it can deal with partial observable, non deterministic domains and asynchronous, message-based interactions between the domain (encoding the component services) and the plan (encoding the composite service). Moreover, the framework can handle complex transactional and termination requirements since it supports a goal language that allows to specify conditions of different strengths and preferences among different (e.g., primary and secondary) requirements. We extended this framework with new techniques and methods to overcome its limitation for what concerns the specification of data flow requirements and the encoding of data knowledge within the composition domain. Clearly, the data flow is as critical for the composition problem as the control flow, since the execution of a service is driven by the received and manipulated data. However, considering data in Web service composition has to deal with several problems: data domains are often infinite, and the semantics of data structures is complex (e.g. service messages are XML documents and service functions are XPath expressions). One of the key contributions is the possibility to handle the complex data flow composition requirements defined thorugh the data net language [6]. Moreover, we extended the framework with the K-level approach [11]: a novel abstraction-based approach for handling data, which ranges over an infinite domain, in a finite, symbolic way.

2.3 WS-Compose

The approach presented in this paper has been implemented as a prototype toolkit, namely WS-Compose, and integrated in the ASTRO Toolset [1], a toolkit providing an integrated environment for the composition of Web services. The ASTRO Toolset covers several aspects of the Web service composition process by providing tools and techniques supporting the analyst in the different phases (e.g. design time verification, run time monitoring, automated composition), and allows for the usage of industrial standards such as WSDL and WS-BPEL in the definition of Web services. For what concerns the automated synthesis of new services, WS-Compose supports all the phases of Web service automated composition: from the specification of control-flow and data-flow requirements by means of graphical tools for drawing data net diagrams and specifying control-flow requirements (WS-Req), to the automatic synthesis of the desired service (WS-Synth), to the deployment, simulation (WS-animator), and execution of the new composite service.

The ASTRO approach has been evaluated on a wide range of experimental domains, including real Web service composition domains. A significant example is the scenario that requires the composition of the Amazon E-Commerce Services and the e-payment service offered by Banks of Monte dei Paschi di Siena Group (MPS) [7]. The goal of the composition is to generate an *e-Bookstore* application that allows to order books and buy them via a secure credit card payment transaction. This composition scenario is particularly challenging since all component services export complex interaction protocols and handle structured data in messages. The following table shows the results of the eBookstore automated composition problem[1].

[1]The composition times have been obtained on a Pentium Centrino 1.6 GHz with 512 Mb RAM of memory running Linux

	Time (sec.)		WS-BPEL
	model construction	composition & emission	complex activities
E-BOOKSTORE	2.7	605.2	177

We distinguish between model construction time (translate the WS-BPEL component services into STS and encode the composition goal) and composition time (synthesize the composition and emit the corresponding executable WS-BPEL). The task of manually encoding and testing the same composition required several hours of work (more or less 20 hours).

The ASTRO approach has thus shown to be applicable also to this real domain providing a first positive answer to the question of the practical applicability of automated composition techniques.

3 Conclusions

Developing composite processes interacting with complex real world web services requires a time consuming analysis of the component services, both for what concerns their interaction protocol and the data structure of their messages. Moreover, it requires a detailed implementation of the new composite service that takes into account all the possible interaction evolutions (faults, exceptions). We propose an automated composition approach that can deal with real world composition problems and that dramatically reduces the effort for the composition by automatically generating both the internal executable composite process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL). Interesting features to be investigated in the future would be to extend the approach in order to handle *peer-to-peer* and *run-time* automated composition problems.

References

[1] ASTRO Project: *Supporting the Composition of Distributed Business Processes* - http://astroproject.org

[2] R. Akkiraju, B. Srivastava, A. Ivan, R. Goodwin, and T. Syeda-Mahmood. *SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition.* Proc. of IEEE International Conference on Web Services (ICWS'06), 2006

[3] D. Ardagna and B. Pernici. *Dynamic web service composition with QoS constraints.* International Journal of Business Process Integration and Management, V.1, N.4, 233-243, 2006

[4] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. *Composition of Services with Nondeterministic Observable Behaviour.* Proc. of International Conference on Service Oriented Computing (ICSOC'05), 2005

[5] R. Hull, M. Benedikt, V. Christophides, and J. Su. *E-Services: A Look Behind the Curtain.* Proc. PODS'03, 2003

[6] A. Marconi, M. Pistore, and P. Traverso. *Specifying Data-Flow Requirements for the Automated Composition of Web Services.* Proc. of Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM06), 2006

[7] A. Marconi, M. Pistore, and P. Traverso. *Automated Web Service Composition at Work: the Amazon/MPS Case Study.* Proc. of IEEE International Conference on Web Services (ICWS07), 2007

[8] A. Marconi, M. Pistore, and P. Traverso. *An Iterative Approach for the Process level Composition of Web Services.* Workshop Proc. of 3rd South-East European Workshop on Formal Methods (SEEFM07), 2007

[9] M. Pistore, P. Traverso, and P. Bertoli. *Automated Composition of Web Services by Planning in Asynchronous Domains.* Proc. of the International Conference on Automated Planning & Scheduling (ICAPS05), 2005

[10] M. Pistore, P. Traverso, and P. Bertoli and A. Marconi. *Automated Synthesis of Composite BPEL4WS Web Services.* Proc. of IEEE International Conference on Web Services (ICWS05), 2005

[11] M. Pistore, A. Marconi, and P. Traverso and P. Bertoli. *Automated Composition of Web Services by Planning at the Knowledge Level.* Proc. of International Joint Conferences on Artificial Intelligence (IJCAI05), 2005

[12] S. McIlraith and S. Son. *Adapting Golog for Composition of Semantic Web Services.* Proc. of the Eighth International Conference on Knowledge Representation and Reasoning (KR'02), 2002

Choreography Modeling and Analysis with Collaboration Diagrams

Tevfik Bultan
University of California, Santa Barbara
bultan@cs.ucsb.edu

Xiang Fu
Hofstra University
xfu2006@gmail.com

1 Introduction

UML collaboration diagrams (called communication diagrams in [8]) provide a convenient visual model for specifying Web Service choreographies. A choreography specifies the desired set of interactions among a set of Web services. We formalize the interactions among Web services as *conversations*, i.e., the sequence of messages exchanged among the services, recorded in the order they are sent. This paper reviews our recent results on the *realizability problem* for choreographies specified as collaboration diagrams [4, 5]. The realizability problem investigates the following question: Is it possible to construct a set of peers that generate exactly the same set of conversations specified by a given choreography? To study this problem, we model a set of Web services (i.e., peers) as a set of communicating finite state machines [3] and we identify a set of sufficient conditions for realizability of a class of collaboration diagrams.

2 Collaboration Diagrams and Conversations

In a collaboration diagram a set of peers communicate via messages. Each message send event has a unique sequence label. A sequence label consists of a (possibly empty) string of letters (which we call the prefix) followed by a numeric part (which we call the sequence number). The numeric ordering of the sequence numbers defines an implicit total ordering among the message send events with the same prefix. For example, event A2 can occur only after the event A1, but B1 and A2 do not have any implicit ordering. It is also possible to explicitly state dependency relationship among events. For example if an event e is marked with "B2,C3/A2" then A2 is the sequence label of e, and the events with sequence labels B2, C3 and A1 must precede e. In a collaboration diagram we use the notion of *message threads* to refer to a set of messages that have the same prefix (and, therefore, are totally ordered) and that can be interleaved arbitrarily with other messages.

As an example, consider the collaboration diagram in Figure 1 for the Purchase Order Handling service described in the BPEL language specification [2]. All the messages in this example are transmitted asynchronously. There are four threads (the main thread, which corresponds to the empty prefix, and the threads with labels A, B and C). The interactions between the Vendor and the Shipping, Scheduling and Invoicing peers are executed concurrently. However, there are some dependencies among these concurrent interactions: *shipType* message should be sent after the *shipReq* message is sent, the *shipSchedule* message should be sent after the *shipInfo* message is sent, and the *orderReply* message should be sent after all the other messages are sent.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This work is supported by NSF grants CCF-0614002 and CCF- 0716095.

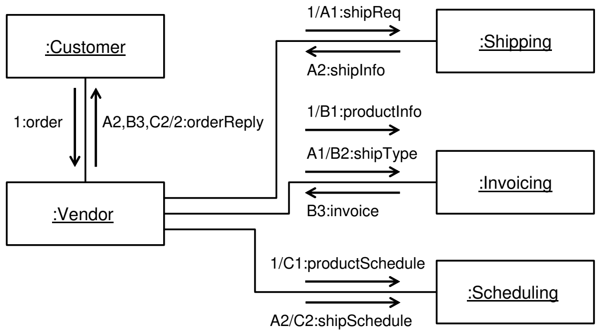


Figure 1: An example collaboration diagram for a composite web service.

Based on the assumptions discussed above we formalize the semantics of collaboration diagrams as follows.

Definition 1: A *collaboration diagram* $\mathcal{D} = (P, L, M, E, D)$ consists of a set of peers P, a set of links $L \in P \times P$, a set of messages M, a set of message send events E, and a dependency relation $D \subseteq E \times E$ among the message send events. Each event has one of the following three recurrence types: 1 (regular), ? (conditional), and $*$ (iterative). A dependency $(e_1, e_2) \in D$ means that e_1 has to occur before e_2. We assume that there are no circular dependencies. An event e is an *initial event* of $\mathcal{D}$ if it has no incoming edges in $\mathcal{D}$.

Given a collaboration diagram $\mathcal{D}$ we denote the *set of conversations* defined by $\mathcal{D}$ as $\mathcal{C}(\mathcal{D})$ where $\mathcal{C}(\mathcal{D}) \subseteq M^*$. $\mathcal{C}(\mathcal{D})$ specifies the desired behaviors in a global perspective. A *conversation* $\sigma = m_1 m_2 \dots m_n$ is in $\mathcal{C}(\mathcal{D})$, i.e., $\sigma \in \mathcal{C}(\mathcal{D})$, if and only if $\sigma \in M^*$ and there exists a corresponding matching sequence of message send events $\gamma = e_1 e_2 \dots e_n$ such that (1) each message in the conversation σ is equal to the message of the matching send event in the event sequence γ; and, (2) the ordering of the events in the event sequence γ does not violate the dependencies in D; and, (3) if an event does not appear in the event sequence γ then it must be either a conditional event or an iterative event; and, (4) only iterative events can be repeated in the event sequence γ.

Next, we model the composition of peers [6, 7]. We assume that each finite state machine has a single FIFO input queue for asynchronous messages. A send event for an asynchronous message appends the message to the end of the input queue of the receiver, and a receive event for an asynchronous message removes the message at the head of the input queue of the receiver.

Definition 2: Each peer $\mathcal{A}_i = (M_i, T_i, s_i, F_i, \delta_i)$ is a nondeterministic FSA where $M_i = M_i^A \cup M_i^S$ is the set of messages that are either received or sent by p_i, T_i is the finite set of states, $s_i \in T$ is the initial state, $F_i \subseteq T$ is the set of final states, and $\delta_i \subseteq T_i \times (\{!, ?\} \times M_i \cup \{\epsilon\}) \times T_i$ is the transition relation. A transition $\tau \in \delta_i$ can be one of the following three types: (1) a send-transition of the form $(t_1, !m, t_2)$, and (2) a receive-transition of the form $(t_1, ?m, t_2)$, and (3) an ϵ-transition of the form (t_1, ϵ, t_2).

A *run* of peers is a sequence of actions (as defined above) taken by the peers. A *complete run* is one such that at the end of run each peer is in a final state and each FIFO queue is empty. The corresponding sequence of messages induced from the send events of a run is called a *conversation*. Given a set of peer state machines $\mathcal{A}_1, \dots, \mathcal{A}_n$ we denote the set of conversations generated by them as $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$. We call a set of peers *well-behaved* if each partial run is a prefix of a complete run (i.e., well-behaved peers never get stuck).

Definition 3: Let $\mathcal{D}$ be a collaboration diagram. We say that the peer state machines $\mathcal{A}_1, \dots, \mathcal{A}_n$ *realize* $\mathcal{D}$ if $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n) = \mathcal{C}(\mathcal{D})$. A collaboration diagram $\mathcal{D}$ is *realizable* if there exists a set of well-behaved peer state machines that realize $\mathcal{D}$.

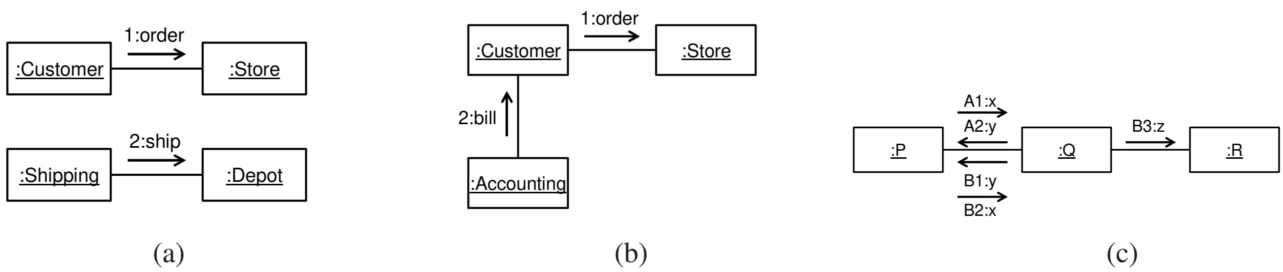


Figure 2: Unrealizable collaboration diagrams.

Not all collaboration diagrams are realizable. For example, Figure 2(a) shows a simple collaboration diagram that is not realizable. The conversation set specified by this collaboration diagram is $\{$*order ship*$\}$, i.e. this collaboration diagram specifies a single conversation in which, first, the Customer has to send the *order* message to the store, and then the Shipping department has to send the *ship* message to the Depot. However, this conversation set cannot be generated by any implementation of these peers. Any set of peer state machines that generates the conversation "*order ship*" will also generate the conversation "*ship order*". The Shipping department has no way of knowing when the *order* message was sent to the Store, so it may send the *ship* message before the *order* message which will generate the conversation "*ship order*". Since the conversation "*ship order*" is not included in the conversation set of the collaboration diagram shown in Figure 2(a), this collaborations diagram is not realizable. Figure 2(b) and (c) show two other collaboration diagrams that are not realizable.

3 Sufficient Conditions for Realizability

In this section we present sufficient conditions for realizability of collaboration diagrams.

Definition 4: We call a collaboration diagram *separated* if each message appears in the event set of only one thread, i.e., given a separated collaboration diagram $\mathcal{D} = (P, L, M, E, D)$ with k threads, the event set E can be partitioned as $E = \bigcup_{i=1}^{k} E_i$ where E_i is the event set for thread i, $M_i = \{e.m \mid e \in E_i\}$ is the set of messages that appear in the event set E_i and $i \neq j \Rightarrow M_i \cap M_j = \emptyset$.

Note that dependencies among the events of different threads are still allowed in separated collaboration diagrams. The collaboration diagrams in Figure 1, Figure 2(a) and (b), are separated whereas the collaboration diagram in Figure 2(c) is not separated (because message x is involved in two threads A and B). Based on our experience, requiring a collaboration diagram to be separated is not a significant restriction in practice.

Definition 5: We call the event e *well-informed* if one of the following conditions hold: (1) e is an initial event. (2) The immediate predecessor of e is either a synchronous message send event, or if it is not a conditional or iterative send event, then for e to be well-informed, the sender of the message for e has to be either the receiver or the sender of the message for its immediate predecessor. (3) If an immediate predecessor of an event e is either a conditional or an iterative asynchronous message send event, then, to be well-informed, e cannot be a conditional or iterative send event and it must have the same sender and the receiver but a different message than its immediate predecessor.

Theorem 6: A separated collaboration diagram $\mathcal{D}$ is realizable if all the events $e \in E$ are well-informed.

The proof of the above property is given in [5]. Note that, the events with label 2 in Figures 2(a) and (b) are not well-informed. Well-informedness of the events alone does not guarantee realizability of a collaboration diagram. Consider the unrealizable and un-separated collaboration diagram shown in Figure 2(c). This collaboration diagram has two threads (A and B) and it is not separated since both threads have send events for messages x and y. Note that, although all the events in this collaboration diagram are well-informed, this collaboration diagram is not realizable. The conversation set specified by this collaboration diagram consists of all interleavings of the sequences xy and yxz which is the set $\{xyyxz, xyxyz, xyxzy, yxzxy, yxxzy, yxxyz, yxyxz\}$. However any set of peer state machines that generate this conversation set will either generate the conversation $xyzxy$ or will not be well-behaved. Consider any set of peer state machines that generate this conversation set. Consider the partial run in which first peer P sends x and then the peer Q sends y. From the peer Q's perspective there is no way to tell if y was sent first or if x was sent first. If we require peer Q to receive the message x before sending y (hence, ensuring that x is sent before y) then we cannot generate the conversations that start with the prefix yx. Hence, peer Q can continue execution assuming that the conversation being generated is $yxzxy$ and send the message z before peer P sends another message. Such a partial execution will generate the sequence xyz which is not the prefix of any conversation in the conversation set of the collaboration diagram. Therefore such a partial execution will either lead to a complete run and generate a conversation that is not allowed or it will not lead to any complete run, either of which violate the realizability condition.

4 Conclusion

To the best of our knowledge, realizability of collaboration diagrams has not been studied before our work in [4, 5]. There were similar efforts on Message Sequence Charts (MSCs) [1]. However, as MSCs concentrate on specification of local behaviors, earlier results on realizability of MSCs are not applicable to the realizability of collaboration diagrams. In our earlier work, we have studied the realizability of conversations specified using automata, called *conversation protocols* [6, 7].

Analysis of interactions specified by collaborations diagrams is becoming increasingly important in the web services domain where autonomous peers interact with each other through messages to achieve a common goal. Since such interactions can cross organizational boundaries, it is necessary to focus on specification of interactions rather then the internal structure of individual peers. We argue that collaboration diagrams are a useful visual formalism for specification of interactions among web services. However, specification of interactions from a global perspective inevitably leads to the realizability problem. Our work formalizes the realizability problem for collaboration diagrams and gives sufficient conditions for realizability.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808, 2001.

[2] Business process execution language for web services (BPEL), version 1.1. `http://www.ibm.com/developerworks/library/ws-bpel`.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[4] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. In *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007)*, pages 122–132, 2007.

[5] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.

[6] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and analysis of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, November 2004.

[7] X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, December 2005.

[8] OMG unified modeling language superstructure, version 2.1.2. `http://ww.uml.org/`, October 2007.

Choreography Design Using WS-BPEL

Oliver Kopp Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart
{kopp,leymann}@iaas.uni-stuttgart.de

Abstract

Web Services are the state-of-the-art realization of a service-oriented architecture. While there is an agreed standard to describe the interface of services (WSDL) as well as an agreed standard to describe the behavior of a single process (WS-BPEL), there is no agreed standard to describe choreographies. In this paper, we give an overview about existing approaches to model choreographies and present one approach based on WS-BPEL in detail.

1 Introduction

The service-oriented architecture (SOA) is an architectural style based on the services paradigm. The most popular realization of the SOA paradigm are Web Services [1]: each service is offered as Web Service. Web Services can be combined to form a business process using the Web Services Business Process Execution Language (WS-BPEL, BPEL for short). A BPEL process is in turn offered as Web Service, which enables recursive composition. Forming business processes out of services is called "orchestration". When multiple processes interact with each other, orchestrations describe the point of view of a single process only. In contrast to orchestrations, choreographies describe the interplay between processes from a global perspective. While orchestrations are well understood, choreographies are an open research field. In this paper, we give an overview about the state-of-the-art in choreography modeling and provide detail insight on a choreography language proposal based on BPEL.

Choreographies are used to capture collaborations between multiple business partners from a global perspective. While most of the published scenarios originate from a top-down approach, another use-case for choreographies is a bottom-up approach: for example, if a company acquires another company, the business processes of both have to be adapted to be able to work together and thus to make use of the synergy effects. Important reasons to design choreographies are acquisitions and merges between companies and the formation of virtual enterprises.

In the following, we use a RosettaNet Partner Interface Process (PIP) to illustrate choreography design. RosettaNet is an industry consortium defining "high-value process scenarios that deliver manufacturing quality data, end-to-end supply chain visibility, and legislative compliance" [12]. The process scenarios are described using interconnection models. In an interconnection model, the behavior of each participant and the messages exchanged are shown. A typical PIP is the PIP 3A1 "Request Quote" defined in RosettaNet Cluster 3 "Order Management". There, a buyer decides whether he needs to place an order. If yes, he specifies his quote request

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

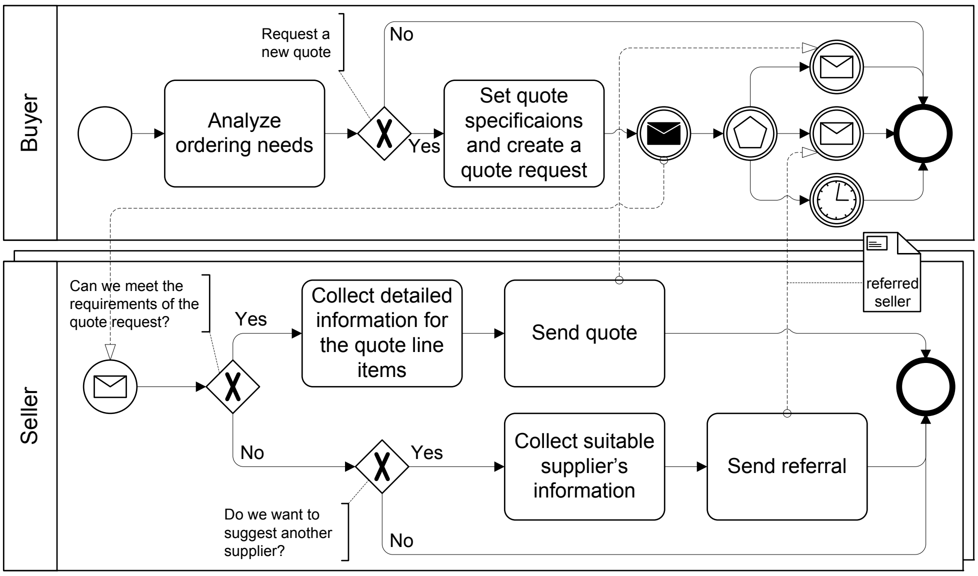


Figure 1: PIP 3A1: Request Quote [13]. Modeled using BPMN with choreography extensions

and sends the quote request to a seller. The seller in turn decides whether he meets the requirements of the quote request. If so, he replies with a quote. If the seller does not meet the requirements, he decides whether he can suggest another supplier. If yes, he sends the suggestion back. If not, he does nothing. Figure 1 presents the BPMN representation of the PIP. We use BPMN V1.1 and the choreography extensions presented in [4]. The shaded pool denotes that there are multiple sellers involved in the choreography. The referenced passed on the message flow is explicitly modeled and associated with the last message flow between the seller and the buyer. In the graphical representation, we assume that each pool is realized by one process. To ensure proper termination of the buyer, we had to include a timeout to handle the case that the supplier does not send any quote and does not send any referral.

In general, in the field of choreography design, there are three issues to tackle: (i) modeling of a choreography, (ii) verification of the choreography and finally (iii) mapping of the choreography to the runtime. In case of choreography modeling, the language to express the choreography has to have well-defined semantics and needs to be suitable to capture choreographies. When a choreography is modeled, the model itself has to be checked for modeling errors: the model has to be consistent within itself (e.g., not contain any deadlock and always reach an end state) and has to fulfill certain constraints (e.g., given by logical formulas). When it comes to execution, the semantics of the choreography has to be captured by local processes, which have to be capable to enact the constraints defined by the choreography.

Currently, there are three main approaches to model choreographies: interaction models, interconnection models and declarative models. Interaction models use the interaction a basic building block. In contrast to interaction models, the main idea of interconnection models is to be close to the execution and to re-use the idea of abstract processes: activities of the local abstract processes are interconnected. An abstract process itself leaves out process internal details, which are not needed to describe the interaction with the partners. While interaction and interconnection models describe all possible interaction schemes, declarative models define constraints on the execution. Thus, declarative models specify the "borders" of possible execution, but do not enumerate explicitly all possible executions [10].

Current languages to specify interaction models are for example the Web Service Choreography Description Language (WS-CDL, [5]) and extensions to BPMN for interaction modeling (iBPMN, [2]). While these languages are suited to capture the interactions between services on a higher level, the runtime-support of them is an open field. The current solution is to map parts of the choreography specification to abstract BPEL process models, which are then refined and executed. However, not all constraints can be directly mapped to BPEL. For example, there is currently no solution to map the blocking wait of WS-CDL to BPEL. In the case of declarative process models, the mapping to BPEL is a complete open research field.

Orchestrations of Web services are mainly defined in BPEL. BPEL has native support for concurrency, backward and forward recovery. To enable modularity and composability, a choreography language should use the same control-flow semantics as an orchestration language to close the gap between choreography specification and runtime. While there is a mapping from BPMN to BPEL available [9], BPMN does not have the expressiveness to specify all the behavior which can be expressed by BPEL constructs. For example, event handlers and termination handlers cannot be modeled using BPMN. Furthermore, a BPEL process can be used to specify the behavior of one participant only. Therefore, we proposed extensions to BPEL to lift BPEL from an orchestration language to a full choreography language (BPEL4Chor [3]). In addition, we added constructs to BPMN to enable modeling choreographies using BPMN including a BPMN representations of BPEL constructs (`http://www.bpel4chor.org/editor`, [11]).

2 BPEL4Chor

BPEL4Chor itself consists of three artifacts: (i) participant behavior descriptions, (ii) a topology description and (iii) a participant grounding.

The *participant behavior descriptions* are abstract BPEL processes describing the behavior of each participant. “Abstract BPEL” denotes that the BPEL processes have to be refined to be fully deployable and to be executed on a BPEL engine. The steps going from an abstract BPEL processes to an executable BPEL process are called “executable completion” and are mainly manual work. It is important to note, that WSDL port types and WSDL operations are not used in the participant behavior descriptions. This allows to specify the behavior of a participant without the fixed connection to concrete realizations. The concrete WSDL information is brought in during the participant grounding.

The BPEL4Chor *topology* provides a global view on the choreography: it defines the participants and the message links. A message link connects communicating activities and corresponds to a message flow in BPMN. The concept of a message link allows to wire existing orchestrations to provide a global view on the interaction.

We see BPEL as orchestration standard and WSDL as standard to describe interfaces. Therefore, the *grounding* brings in the necessary WSDL information to enact the choreography. This information can then be used to generate abstract BPEL containing partner links, port types and operations. These BPEL processes can then serve as basis for the executable completion. However, it is not necessary to implement a participant using BPEL. A participant can also be realized by one or more Web services implemented in any language as long as the behavior of these Web services corresponds to the given participant behavior description.

A BPEL4Chor choreography can be verified using an approached based on Petri nets presented in [8]. There, the choreography’s participants are translated into Petri nets. These nets are then connected according to the BPEL4Chor choreography. If there are multiple participants involved, the respective net is copied accordingly to reflect the multiple instances. The resulting Petri net can be checked for deadlocks or any other desired property using model checking tools. Experiments showed that choreographies with up to thousand instances can be verified [8]. In case a deadlock is found in the choreography, the faulty participant can be fixed automatically [7]. All results of the verification (e.g., deadlock traces) can be mapped back to the original BPEL processes. This allows for a seamless integration of choreography verification into the process of choreography modeling.

If an executable BPEL process was modeled based on a participant behavior description, it has to be checked, whether the executable process conforms to the participant behavior description. A general approach to check conformance of BPEL processes is presented in [6].

3 Summary

We presented an overview of choreography design and BPEL4Chor. We showed how existing technologies can be re-used to describe a choreography: BPEL is used to define the participant behavior descriptions and WSDL is brought in at the grounding to enable the message exchange via an Enterprise Service Bus. The BPEL4Chor topology is the first proposal enabling interconnection of BPEL activities.

BPEL4Chor is part of the Tools4BPEL project and is funded by German Federal Ministry of Education and Research (project number 01ISE08). The other partners involved are the Humboldt-Universität zu Berlin and the MEGA International GmbH. In the project, our task is to investigate the modeling of sub-processes, choreographies, cross-partner fault handling, cross-partner transactions and sub-processes using BPEL. The part of the Humboldt-Universität zu Berlin is to provide verification mechanisms and tools for BPEL as well as for our extensions of BPEL. Finally, MEGA delivers challenging examples guiding and driving our research.

References

[1] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, 2005.

[2] G. Decker and A. P. Barros. Interaction Modeling Using BPMN. In *1st International Workshop on Collaborative Business Processes (CBP)*. Springer, 2007.

[3] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS*. IEEE Computer Society, 2007.

[4] G. Decker and F. Puhlmann. Extending BPMN for Modeling Complex Choreographies. In *CoopIS*. Springer, 2007.

[5] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation, W3C, November 2005.

[6] D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the Compatibility Notion for Abstract WS-BPEL Processes. In *International Conference on World Wide Web*. ACM, 2008.

[7] N. Lohmann. Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In *BPM*. Springer, 2008.

[8] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig. Analyzing BPEL4Chor: Verification and Participant Synthesis. In *WS-FM*. Springer, 2007.

[9] C. Ouyang, M. Dumas, S. Breutel, and A. H. M. ter Hofstede. Translating Standard Process Models to BPEL. In *CAiSE*. Springer, 2006.

[10] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-based workflow models: Change made easy. In *CoopIS*, 2007.

[11] K. Pfitzner, G. Decker, O. Kopp, and F. Leymann. Web Service Choreography Configurations for BPMN. In *WESOA*. Springer, 2007.

[12] RosettaNet. Home page. `http://www.rosettanet.org/`.

[13] RosettaNet. *Overview: Clusters, Segments, and PIPs*. Version 02.04.00.

WAVE: Automatic Verification of Data-Driven Web Services*

Alin Deutsch Victor Vianu
Department of Computer Science & Engineering
University of California, San Diego

Abstract

Data-driven Web services, viewed broadly as interactive systems available on the Web for users and programs, provide the backbone for increasingly complex Web applications. While this yields ever-increasing functionality, the added complexity renders such applications more vulnerable to bugs and failures, potentially compromising their robustness and correctness. Therefore, there is a need to develop verification techniques for such Web services. The WAVE *project at UC San Diego aims to develop new approaches for automatic verification of data-driven Web services. The work relies on a novel, highly effective marriage of model checking and database techniques. We summarize briefly the main contributions of the project, which range from theoretical foundations to the successful implementation of a prototype verifier.*

1 Verification of stand-alone data-driven Web services

We first outline our results on verification of data-driven Web services for single peers in isolation, then dicsuss extensions of the results to compositions of Web services. We focus on services interacting with external users or programs through a Web browser interface, and accessing an underlying database. Such services include e-commerce sites, scientific and other domain-specific portals, e-government, etc. These Web sites are often governed by complex, data-dependent workflows, controlled by queries. The spread of such services has been accompanied by the emergence of tools for their high-level specification. A representative, commercially successful example is WebML [1], which allows to specify a Web application using an interactive variant of the E-R model augmented with a workflow formalism. The code for the Web application is automatically generated from the WebML specification. This not only allows fast prototyping and improves programmer productivity but also provides new opportunities for automatic verification. Indeed, our WAVE prototype automatically verifies a significant class of such services. Verification leads to increased confidence in the correctness of database-driven Web applications generated from high-level specifications, by addressing the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation).

We focus on interactive Web sites generating Web pages dynamically by queries on an underlying database. The Web site accepts input from external users or programs, possibly subject to specified pre-conditions. It responds by taking some action, updating its internal state database, and moving to a new Web page determined

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Supported by the NSF under grant numbers IIS/0415257 and CAREER/0347968. Address: CSE 0404, UC San Diego, La Jolla, CA 92093-0404, USA

by yet another query. We model the queries used in the specification of the Web service as first-order queries (FO), also known as relational calculus, which can be viewed as an abstraction of the data manipulation core of SQL. A *run* is a sequence of inputs together with the Web pages, states, and actions generated by the Web service. The properties we wish to verify range from basic soundness of the specification (e.g. the next Web page to be displayed is always uniquely defined) to semantic properties (e.g. no order is shipped before a payment in the right amount is received). Such properties are expressed using an extension of *linear-time temporal logic* (LTL). Recall that LTL is propositional logic augmented with temporal operators such as **always**, **eventually, next** and **until**. The extension uses FO formulas in place of the atomic propositions of classical LTL, yielding a language called LTL-FO.

For example, the following is an LTL-FO formula stating that if a product x is paid at some point in the right amount y, then x is eventually delivered:

$$\forall x \forall y\ \ \mathbf{always}[(pay(x,y) \wedge price(x,y)) \rightarrow \mathbf{eventually}\ (deliver(x))]$$

Here *pay* is an input, *price* is a database relation, and *deliver* is an action relation.

The task of a verifier is to check that all runs of the Web service satisfy a given LTL-FO property (as usual in verification, runs are considered to be infinite). Verifiers search for counter-examples to the desired property, i.e. runs leading to a violation. A verifier is *complete* if it is guaranteed to find a counter-example whenever one exists. In the broader context of verification, a database-driven Web service is an *infinite-state* system, because the underlying database queried by the application is not fixed in advance. This poses an immediate and seemingly insurmountable challenge. Classical verification deals with finite-state systems, modeled in terms of propositions. For more expressive specifications, the traditional approach suggests the following strategy: first abstract the specification to a fully propositional one and next apply an existing model checker such as SPIN [6] to verify LTL properties of the abstracted model. This approach is unsatisfactory when the data values are first-class citizens, as in data-driven Web applications. For example, abstraction would allow checking that *some* product was delivered after *some* payment was completed. However, we could not inspect the payment and product data values to verify that the payment was for the delivered item, and in the correct amount. Conventional wisdom holds that, short of using abstraction, it is hopeless to attempt complete verification of infinite-state systems. In this respect, WAVE represents a significant departure because it is complete for a practically relevant class of infinite-state specifications. As far as we know, this is the first implementation of such a verifier.

In general, complete verification is easlily seen to be undecidable. Thus, completeness is only guaranteed under certain restrictions described shortly. To show that these restrictions cover a large class of applications, we have modeled a computer shopping Web site similar to the Dell site, an airline reservation application similar to Expedia, an online bookstore in the spirit of Barnes & Noble, and a sports Web site on the Motorcycle Grand Prix. We used these applications in our experimental evaluation of WAVE. If the specification and the property do not satisfy the restrictions needed for completeness, WAVE can still be used as an incomplete verifier, as typically done in software verification. The heuristics we developed remain just as effective in this case.

We now describe informally the restrictions on the Web service specifications and properties that guarantee completeness, called *input boundedness* [7, 5]. Recall that the queries we use in the specification of Web service as well as properties are FO queries. In a nutshell, input boundedness restricts the range of quantifications in FO formulas to values occurring in the input. This is natural, since interactive Web applications are input-driven. For example, to state that every payment received is in the right amount, one might use the input-bounded formula $\forall x \forall y[pay(x,y) \rightarrow price(x,y)]$, where $pay(x,y)$ is an input and $price$ is a database relation providing the price for each item.

Our main theoretical result shows the decidability of model checking for input-bounded specifications and properties. The complexity of checking that a Web service specification $\mathcal{W}$ satisfies an LTL-FO property φ is shown to be PSPACE. We briefly describe the technique underlying this result, as well as the implementation of WAVE. In our scenario, a first difficulty facing a verifier is that exhaustive exploration of all possible runs of a

Web service $\mathcal{W}$ on all databases is impossible since there are infinitely many possible databases and the length of runs is infinite. The solution lies in avoiding explicit exploration of the state space. Instead of materializing a full initial database and exploring the possible runs on it, we generate a compact representation of equivalence classes of actual runs, called *pseudo-runs*, by lazily making at each point in the run just the assumptions needed to obtain the next configuration and check satisfaction of φ. Specifically, for input-bounded $\mathcal{W}$ and φ, this can be done as follows:

(i) explicitly specify the tuples in the database that use only a small set of relevant constants C computed from $\mathcal{W}$ and φ; this is called the *core* of the database and remains unchanged throughout the run. Its size is polynomial in $\mathcal{W}$ and φ.

(ii) at each step in the run, make additional assumptions about the content of the database, needed to determine the next possible configurations. The assumptions involve only a small set of additional values.

The key point is that the local assumptions made in (ii) at each step need not be checked for global consistency. Indeed, a non-obvious consequence of the input-bounded restriction is that these assumptions are guaranteed to be globally consistent with *some* very large database which is however never explicitly constructed. Since pseudo-run configurations are of polynomial size, this yields a PSPACE verification algorithm and establishes our main theoretical result [5].

Theorem 1: Given an input-bounded Web service specification $\mathcal{W}$ and LTL-FO formula φ, it is PSPACE-complete whether $\mathcal{W}$ satisfies φ.

The PSPACE upper bound holds assuming a fixed bound on the arity of database and state relations. Otherwise, the complexity is EXPSPACE (with the arity in the exponent). It is worth noting that, in the broader context of static analysis, the PSPACE complexity is the best one can hope for. Indeed, recall that even satisfaction of a propositional LTL property by a finite-state Mealy machine is already PSPACE-complete.

The input-boundedness restriction imposed for decidability turns out to be quite tight. Indeed, we showed that even minor relaxations to these restrictions lead to undecidability. Some extensions to the model also lead to undecidability, such as allowing key constraints on the database. On the other hand, PSPACE decidability continues to hold with built-in predicates such as a dense order on the domain.

The WAVE verifier To explore the practical feasibility of our ideas, we embarked upon the implementation of the WAVE verifier. First, we developed a tool for high-level, efficient specification of data-driven Web services, in the spirit of WebML. Next, we implemented WAVE taking as input a specification of a Web service using our tool, and an LTL-FO property to be verified. The implementation is made possible by a novel coupling of classical model-checking with database optimization techniques. Interestingly, the starting point is the pseudo-run technique used to show the PSPACE upper bound. However, verification becomes practical only in conjunction with an array of additional heuristics and optimization techniques, yielding critical improvements. Chief among these is dataflow analysis, allowing to dramatically cut down the number of database cores and pseudo-runs generated in a search.

We evaluated the verifier on a set of practically significant Web application specifications, mimicking the core features of sites such as Dell, Expedia, and Barnes and Noble. The experimental results are quite exciting: we obtained surprisingly good verification times (on the order of seconds), suggesting that automatic verification is practically feasible for large classes of properties and Web services. We describe the implementation and our experimental results in [2]. A demo of the WAVE prototype is presented in [4] and is also available at http://db.ucsd.edu/wave.

2 Extension to Web service compositions

The above results apply to the verification of single peers in isolation. We extended these results to the more challenging but practically interesting case of *compositions* of Web services. Asynchronous communication between peers adds another dimension that has to be taken into account. We briefly describe the model and results.

In a composition of Web services, peers communicate with each other by sending and receiving messages via one-way channels implemented by *message queues*. Each queue is associated with a unique sender who places messages into the queue, and a unique receiver who consumes messages from it in FIFO order (thus, we assume messages arrive in the same order they were sent). The messages can be *flat* or *nested*. Flat messages consist of single tuples, e.g. the age and social security number of a given customer. Nested messages consist of a set of tuples, e.g. the set of books written by an author.

As in the stand-alone case, each peer can receive external inputs and produce actions (sets of tuples). In a composition, each peer additionally consumes messages from its input queues, and generates output messages. A *configuration* of the composition consists of the configurations of all participating peers (the database, their local state relations, inputs, current action relations, and the message queues). A run of the composition is a sequence of consecutive configurations. We only consider serialized runs, in which at every step precisely one peer performs a transition. Properties of runs to be verified are specified in an extension of LTL-FO, where the FO statements may additionally refer to the messages currently read and sent.

In order to obtain decidability of verification, we need to extend the input-boundedness restriction introduced for single peers. Naturally, we need to also require input-boundedness of the queries defining output messages. Additional restrictions must be placed on the message channels: they may be lossy, but are required to be bounded. With these restrictions, verification is again shown to be PSPACE-complete (for fixed-arity relations, and EXPSPACE otherwise).

The above model of compositions assumes that all specifications of participating peers are available to the verifier. However, compositions may also involve autonomous parties unwilling to disclose the internal implementation details. In this case, the only information available is typically a specification of their input-output behavior. This led us to investigate *modular* verification. This consists in verifying that a subset of fully specified peers behaves correctly, subject to input-output properties of the other peers. We obtained similar decidability results for verification, subject to an appropriate extension of the input-boundedness restriction.

The results on verification of Web service compositions are described in [3].

Conclusion The results of the WAVE project obtained so far are very encouraging. They suggest that interactive applications controlled by database queries may be unusually well suited to automatic verification, and that our approach based on a mix of model checking and database optimization techniques may come to have significant practical impact.

References

[1] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing data-intensive Web applications*. Morgan-Kaufmann, 2002

[2] A. Deutsch, M. Marcus, L. Sui, V. Vianu and D. Zhou: A verifier for interactive, data-driven Web applications. *ACM SIGMOD Conference* 2005: 539-550

[3] A. Deutsch, L. Sui, V. Vianu, D. Zhou: Verification of communicating data-driven Web services. *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2006: 90-99

[4] A. Deutsch, L. Sui, V. Vianu, D. Zhou: A system for specification and verification of interactive, data-driven Web applications (demo paper). *ACM SIGMOD Conference* 2006: 772-774

[5] A. Deutsch, L. Sui, V. Vianu: Specification and verification of data-driven Web applications. Invited to special issue of *J. Comput. Syst. Sci.* 73(3): 442-474 (2007). Extended abstract in *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2004: 71-82

[6] G. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley, 2003

[7] M. Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.* 66(1):40–65 (2003). Extended abstract in *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2000: 92-103

Web Service Protocols: Compatibility and Adaptation

Marlon Dumas
University of Tartu, Estonia
marlon.dumas@ut.ee

Boualem Benatallah, Hamid R. Motahari Nezhad
The University of New South Wales, Australia
{boualem,hamidm}@cse.unsw.edu.au

Abstract

This paper discusses the notion of protocol compatibility between Web services, and reviews a number of techniques for detecting incompatibilities and for synthesizing adapters for otherwise incompatible services. The paper also reviews related notions such as realizability, substitutability and controllability.

1 Introduction

The composition of Web services involves wiring together multiple web services and having them interact often in ways not originally foreseen during their initial development. In doing so, it is unavoidable that incompatibilities may arise and need to be identified and resolved. We classify these incompatibilities into two types: (i) *signature incompatibilities* that arise when a service requires an operation from another service, but this latter service does not offer it, or when a service A needs to exchange a message with another service B, but the schema of the message that A produces is not compatible with the one that B expects; and (ii) *protocol incompatibilities* that arise when a service A engages in a series of interactions with another service B, but the order in which service A undertakes these interactions is not compatible with that of B.

This paper discusses the notion of protocol compatibility between Web services and reviews a number of techniques for detecting incompatibilities and for synthesizing adapters for otherwise incompatible services. The paper also reviews related notions such as realizability [6], substitutability [4] and controllability [10].

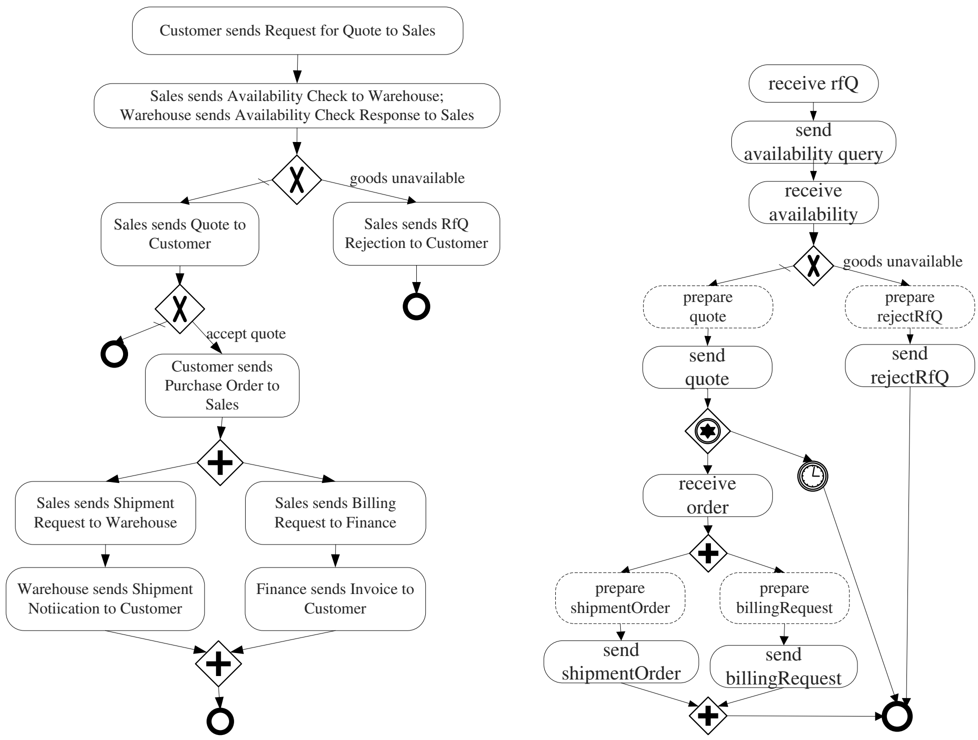
The next section introduces background concepts for modeling web service interactions in general, and service protocols in particular. Section 3 introduces the notion of protocol compatibility and related concepts. Section 4 discusses techniques for synthesizing adapters for protocol-incompatible services. Finally, Section 5 summarizes the discussion and raises directions for future work.

2 Service Interaction Modeling

It is customary to distinguish between two types of models of service interactions: choreographies and orchestrations [15]. A choreography describes interactions between a collection of services from a global perspective. In a choreography, no service plays a privileged role. Figure 1(a) depicts a choreography in the Business Process Modeling Notation (BPMN) [14]. Four services are involved in this choreography: customer, sales, warehouse and finance. Each activity denotes an interaction between two services. Importantly, a choreography only shows interactions, as opposed to actions performed internally by a service. In contrast, an orchestration describes the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

interactions between a designated service (the orchestrator) and a plurality of subordinated services. Figure 1(b) depicts an orchestration for the sales service. An orchestration may include internal actions or timeouts. For example, Figure 1(b) includes four actions internal to the sales service (the four "prepare" actions in dashed lines) and a timeout: After sending a quote, the sales service waits for an order until the quote's expiry time.



(a) Order management choreography (b) Orchestration for Sales service

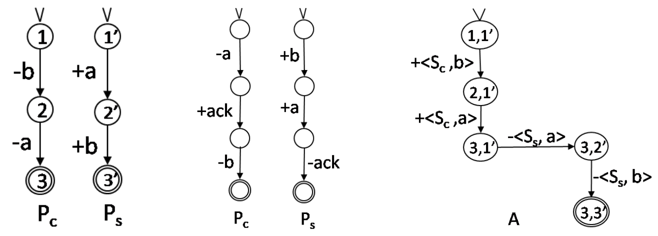
Figure 1: Examples: Choreography and Orchestration

If we consider a multi-party choreography and restrict it to those interactions that involve a given pair of services – e.g. the interactions between the sales and the customer services in the above example – we obtain a (bilateral) *service protocol*. Service protocols described from the perspective of one participant are also called *behavioral interfaces* [7], because they define the behaviour of a service vis-a-vis of one of its clients or peers.

The derivation of behavioural interfaces from choreographies may require refinements. Consider, e.g. the choice in Figure 1(a) that the customer performs between accepting or rejecting a quote. If the customer accepts the quote, it sends an order. Thus, when receiving an order the sales service knows that the customer accepted the quote. However, if the customer rejects the quote, it does not send any message. When deriving a behavioural interface for the sales service, one needs to insert either a timeout (as in Figure 1(b)) or an additional interaction through which the customer communicates the rejection to the sales service. Otherwise, the sales service will wait indefinitely for an order. The notion of *realizability* [6] (also called enforceability [18]) captures this issue. A choreography is *realizable* if the behavioural interfaces obtained by projection of the choreography into each of its participating *roles*, collectively enforce all control-flow constraints in the choreography.

Languages for specifying choreographies, protocols and orchestrations include BPMN (see above) and BPEL.[1] In BPEL, orchestrations are defined down to the point where they can be executed by dedicated platforms. Also, BPEL allows one to specify protocols/behavioral interfaces. For formal analysis, protocols may be represented using e.g. finite state machines (FSMs) [2], process algebra [11] or Petri nets [10, 3].

[1] `http://www.oasis-open.org/committees/wsbpel/`



(a) (b) (c)

Figure 2: Incompatibilities & adaptation: (a) unspecified reception, (b) deadlock, (c) adapter for protocols in (a)

3 Compatibility

Two services are protocol-compatible if every joint execution of these services leads to a proper final state, i.e. a state in which both services are in a final state in their respective protocols [2]. Under the assumption synchronous communication, Yellin & Ström [17] identify two main types of protocol mismatches: *unspecified reception*, in which one party sends a message while the other is not expecting it; and *deadlock*, the case where both parties are mutually waiting to receive some message from the other. To illustrate the concepts, consider the protocols of P_s (of service S_s) and P_c (of service S_c) in Figure 2(a): P_c sends message b (shown by a -b), while P_s does not expect to receive it (unspecified reception). In Figure 2(b) instead, P_c expects to receive message ack after sending a (shown by +ack), while P_s is waiting to receive b (+b). This is a deadlock case. Two protocols are said to be *compatible* if they have no unspecified receptions and they are deadlock-free.

The protocol A' obtained by reversing the polarity of every message in a protocol A is called the mirror of A. In other words, sent messages in A become received messages in its mirror A', while received messages become sent messages. In general, a service protocol is compatible with its mirror. However, if a protocol specification includes internal actions (e.g. timers or evaluation of boolean conditions resulting in certain branches being taken) it is possible that this protocol is not compatible with its mirror protocol, nor with any other protocol. If so, the service is said to be uncontrollable [10]. The problem of controllability is intuitively related to that of realizability – as that they both result when internal choices are not externalized as messages. However, a formal relation between controllability and realizability is yet to be established.

Replaceability (or *substitutability*) refers to the ability for a service to replace another one without inducing incompatibilities [4]. In ServiceMosaic [2], two main classes of replaceability are defined: *subsubmption* and *equivalence*. Protocol P_1 *subsumes* P_2 if P_1 supports at least all the execution traces that P_2 supports. If so, a service S_1 (with protocol P_1) can replace service S_2 (with protocol P_2). If P_1 subsumes P_2, and P_2 subsumes P_1, then P_1 and P_2 are equivalent, and services S_1 and S_2 can be used interchangeably. Finer notions of replaceability are defined in terms of bisimulation [3].

Finally, one can ask the question of whether an orchestration *conforms to* a protocol. If we take the orchestration and we project it to those interactions that appear in its protocol, the question is whether or not the projected orchestration is compatible with the service's protocol. This question is studied in [9].

4 Adaptation

When two services are incompatible, it may be possible to introduce an adapter to resolve their mismatches. In such cases, the service protocols are said to be *adaptable*. Depending on the types of mismatches, it may be possible to automatically synthesize an adapter. The question of synthesizing adapters for incompatible protocols has been studied in the area of SOA, as well as earlier in the area of component-based software engineering.

Yellin & Ström [17] propose an approach for checking the existence of an adapter for incompatible protocols. An adapter is modelled as an FSM consisting of a set of states, a set of typed memory cells to store the messages received by the adapter, and a set of state transition rules. Each rule describes a transition from a state to another in the adapter based on sending or receiving messages, along with a set of memory actions that store or retrieve messages in/from the cells. A rule also constructs messages that need to be sent to partners. The adapter's protocol is said to be compatible with protocols P_1 and P_2 of the adapted components, if their interactions have no unspecified reception and are deadlock free. Figure 2(c) shows an example of an adapter for protocols in Figure 2(a). To synthesize the adapter specification for a pair of components, their interface mappings is required as the input (e.g. which messages should be mapped to which other messages). The adapter synthesis process explores all possible interactions between the protocols P_1 and P_2 and adds them to the adapter protocol. If there are states leading to deadlocks or with unspecified reception, they are removed from the adapter protocol.

Other proposals rely on alternative protocol specification languages that explicitly support concurrency. Mateescu et al. [11] propose a technique for adapter synthesis based on protocols specified using process algebra. Similarly, Brogi et al. [5] provide an automated adapter synthesis approach for protocols specified in BPEL.

Another line of research for service adapter development proposes to characterise the classes of possible mismatches between protocols, provides guidelines for users to identify them and proposes templates to resolve mistmatches based on design patterns [1] or composable adaptation operations [8]. In these approaches, the construction of adapters requires manual intervention. Some of these approaches, e.g. [8] deal with mismatch patterns not supported in automated approaches – e.g. mismatches where a message emitted by a service needs to be mapped to an unbounded number of messages in the receiving service.

Automated approaches for adapter generation make the following assumptions: (i) there is no mismatch at the interface-level, or the correct interface mappings have been provided as the input, and (ii) if there are interactions which lead to deadlocks, they are not adaptable. As discussed in [12], the interface-level mappings can not be always correctly identified without considering the protocol specifications. Second, some deadlock cases may be adaptable, e.g. the resolution of a deadlock may require the generation of messages (e.g., an acknowledgment) that can be constructed in the adapter via user-defined functions.

To address these limitations, Motahari Nezhad et al. [12] approach adapter development as an iterative process consisting of both interface-level and protocol-level mismatch identification and resolution. Their approach starts from an initial set of interface matchings, computed by matching the WSDL interfaces of services, and then, considering the protocol specifications of two services, identifies all the interactions that results in deadlocks. The result is presented in the form of a *mismatch tree* to the user, where the user can identify if such interactions are resolvable. The approach also helps the user by analyzing the mismatch tree. Some deadlock cases may be handled by going back to the interface matching step and refining the interface matchings.

5 Summary and outlook

Figure 3 summarizes the notions introduced in the paper. This panorama summarizes a significant body of research work in the area of service-oriented computing. In this body of work, research questions are often approached under the assumption that the choreographies, protocols and/or orchestrations are known and given as input. Sometimes however, these specifications are unavailable or they are incompletely or unreliably specified, yet one needs to make assertions regarding the correct behavior of a service-oriented system. Recent work has addressed the question of analyzing logs representing the observed behavior of a service-oriented system in order to determine if these logs conform to a choreography or protocol specification [16]. One of the key issues in this setting is that of "correlation", that is, how to group together log entries (such as those in message logs) to produce trails that represent conversations between two or more services [13]. Open questions in this area include investigating the application of techniques from machine learning and information clustering.

An open question in the field of service adaptation is how to maintain adapters in an environment where services evolve continuously. For example, given two services S1 and S2 that communicate through an adapter, how can this adapter be updated (with minimal effort) when either S1 or S2 evolve or are replaced?

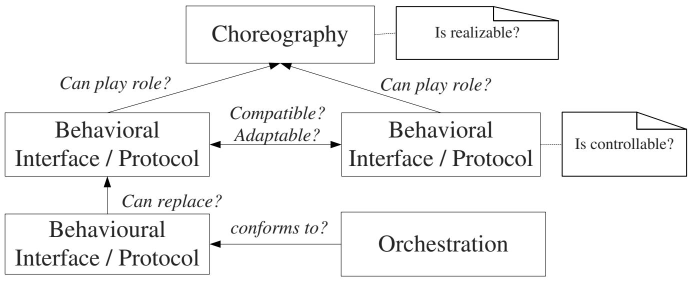


Figure 3: Relations between service interaction modeling viewpoints

References

[1] B. Benatallah, F. Casati, D. Grigori, H. Motahari Nezhad, and F. Toumani. Developing Adapters for Web Services Integration. In *Proc. of CAiSE*, pages 415–429, 2005.

[2] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3):327–357, 2006.

[3] F. Bonchi, A. Brogi, S. Corfini, and F. Gadducci. Compositional Specification of Web Services Via Behavioural Equivalence of Nets: A Case Study. In *Proc. of PETRI NETS*, pages 52-71, 2008.

[4] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In *Proceedings of the 5th International Workshop on Technologies for E-Services (TES)*, pages 15–28, 2004.

[5] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC*, 2006.

[6] T. Bultan, X. Fu, and J. Su. Analyzing conversations: Realizability, synchronizability, and verification. In L. Baresi and E. D. Nitto, editors, *Test and Analysis of Web Services*, pages 57–85. Springer, 2007.

[7] R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.

[8] M. Dumas, M. Spork, and K. Wang. Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In *Proc. of BPM*, pages 65–80, 2006.

[9] D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the compatibility notion for abstract WS-BPEL processes. In *Proc of WWW*, pages 785–794, May 2008.

[10] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.

[11] R. Mateescu, P. Poizat, and G. Salaün. Behavioral adaptation of component compositions based on process algebra encodings. In *Proc. of ASE*, pages 385–388, 2007.

[12] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proc. of WWW*, pages 993–1002, 2007.

[13] H. R. Motahari Nezhad, R. Saint-Paul, B. Benatallah, F. Casati, and P. Andritsos. Process spaceship: Discovering and exploring process views from event logs in data spaces. In *Proc. of VLDB*, 2008.

[14] Object Management Group. Business Process Modeling Notation, V1.1. *OMG Available Specification*, January 2008.

[15] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.

[16] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Techn.*, 8(3), 2008.

[17] D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):292–333, 1997.

[18] J. M. Zaha, M. Dumas, A. H. M. ter Hofstede, A. P. Barros, and G. Decker. Service interaction modeling: Bridging global and local views. In *Proc. of EDOC*, 2006.

Process Mining in Web Services: The WebSphere Case

W.M.P. van der Aalst and H.M.W. Verbeek
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst,h.m.w.verbeek}@tue.nl

Abstract

Process mining has emerged as a way to discover or check the conformance of processes based on event logs. This enables organizations to learn from processes as they really take place. Since web services are distributed over autonomous parties, it is vital to monitor the correct execution of processes. Fortunately, the "web services stack" assists in collecting structured event logs. This information can be used to extract new information about service processes (e.g., bottlenecks, unused paths, etc.) and to check the conformance (e.g., deviations from some predefined process). In this paper, we illustrate the potential of process mining in the context of web services. In particular, we show what a process mining tool like ProM can contribute in IBM's WebSphere environment.

1 Introduction

In a *Service Oriented Architecture* (SOA) services are interacting by exchanging messages, and by combining services more complex services are created. *Choreography* is concerned with the composition of such services seen from a global viewpoint focusing on the common and complementary observable behavior. Choreography is particularly relevant in a setting where there is not a single coordinator. *Orchestration* is concerned with the composition of such services seen from the viewpoint of single service. Independent of the viewpoint (choreography or orchestration) there is a need to make sure that the services work together to ensure the correct execution of business processes.

This paper explores the use of *process mining* [1] in the context of IBM's WebSphere product. WebSphere provides a state-of-the-art infrastructure for realizing a SOA and supports elaborate logging facilities [2]. The Common Event Infrastructure (CEI) offers a systematic way of recording events. Using this information, we can apply the many process mining techniques provided by the process mining tool *ProM* [4].

CEI provides facilities for the generation, propagation, persistence, and consumption of events. Events are represented using the Common Base Event (CBE) model, a standard XML-based format defining the structure of events. For many applications, the information stored in CEI may be too large. Hence, CEI is often only used as a transport layer and events are removed, filtered, or aggregated by IBM tools such as the WebSphere Business Monitor. (But also others such as the Web Services Navigator [3].)

The WebSphere Business Monitor [2] measures the performance of a process based on key performance indicators (KPIs) and the business metrics. Performance related results are displayed in dashboards and used

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

as reference for redesign. The monitoring and analysis tools are not able to discover causal relations between tasks or employees involved in the process, and, thus, they can not extract a process model from the event log. Moreover, an audit of the process to see if it conforms to the organizational procedures and regulations is hardly objective or efficient without having a good understanding of the real process.

This paper demonstrates that process mining is possible and valuable in a SOA context, using WebSphere as an example. However, our findings are quite general and can be applied to other platforms (e.g., using Oracle BPEL). The remainder is organized as follows. First, we discuss the requirements for process mining. Then, we provide insight into the analysis results that can be provided by process mining. Finally, we discuss in what way process mining tools extend capabilities of existing monitoring tools.

2 Getting Data: Correlation is Key!

More and more processes leave their "trail" in the form of event logs. Process mining techniques can use these logs in various ways, e.g., to discover the way that people/services really work, to find out if and where this way deviates from the planned process, to support people in performing their duties, and to improve the performance of processes. In order to do this, process mining techniques expect the logs to contain certain information. Therefore, we first elaborate on this information.

For every *process instance* (often referred to as *case*), a sequence of events is recorded. Examples of process instances are customer orders in a order handling process, patients in a treatment process, and applicants in a selection process. It is crucial to note that events need to be linked to process instances in order to enable process mining. The sequence of events describing a single process instance is called a *trace*. An *event log* is a set of process instances. An event may have various properties such as the associated time, the associated activity, the associated data, the person, organization, or software component responsible for emitting the event, and the associated transaction type (e.g., start, complete, and abort). Process mining assumes that each event is associated to an activity. All other event properties are optional, but can be exploited when present.

One of the major challenges in processing the collected data is to link events to process instances. This corresponds to the notion of *correlation*. For example, when tapping of a message exchanged between two services it is crucial to link this message to a particular process instance. In some cases this may be trivial, e.g., when using a workflow engine with a clear process instance concept or when there is a natural global identifier such as the patient id in hospital processes. In other cases, this may be very difficult. For example, in the context of an ERP system like SAP R/3 it is surprisingly difficult to correlate events. For example, events related to a customer order may refer to order line items rather than the customer order or the supplier and customer may use different keys.

To make things more concrete, we now focus on event logging in the context of WebSphere. IBM uses the so-called *Common Event Infrastructure* (CEI) to record, distribute, and manage events. IBM encourages clients to use the following four subsystems: (1) *WebSphere Business Modeler* to design business processes and to identify the things to be measured and analyzed at run-time, (2) *WebSphere Integration Developer* to translate business process models into actual executable code, (3) *WebSphere Process Server* to enact the configured processes, and (4) *WebSphere Business Monitor* to observe the processes, to measure Key Performance Indicators (KPIs), generate reports, show management dashboards, etc. Although these subsystems are connected, they can also be used independently. For example, the WebSphere Business Monitor can also be used in conjunction to other products such as FileNet P8 BPM, etc.

Correlation is important in both the Process Server and the Business Monitor. To execute processes, incoming events (e.g., messages) need to be routed to the corresponding BPEL process instances. For monitoring, it is also important to correlate events. Take, for example, a KPI that measures the average throughput time of a case. Clearly, to be able to measure such a KPI, it is necessary to correlate the events. The WebSphere Business Monitor uses the concept of "monitoring context" to define a container where all events related to the same instance are brought together. It is also interesting to note that both CEI and the WebSphere Business Monitor use concepts such as times associated to events, etc.

We can summarize the above as follows. For process mining, events need to be correlated to process instances. Correlation problems may inhibit the application of process mining. However, as illustrated using the WebSphere suite, correlation is a foundational concept in the development of web services.

3 Analysis using Process Mining

The goal of process mining is to discover, monitor, and improve real processes by extracting knowledge from event logs. Clearly, process mining is particularly relevant in a setting where the actors involved are autonomous and can deviate or have emerging behavior. The more ways in which services, people, and organizations can deviate, the more interesting it is to observe and analyze processes as they are executed.

Three basic types of process mining can be identified:

- **Discovery**: There is no a-priori model, i.e., based on an event log some model is constructed. For example, using the well-known α-algorithm a process model can be discovered based on low-level events.
- **Conformance**: There is an a-priori model. This model is used to check if reality conforms to the model. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.
- **Extension**: There is an a-priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model. An example is the extension of a process model with performance data, i.e., some a-priori process model dynamically annotated with performance data (e.g., bottlenecks are shown by coloring parts of the process model).

In the context of web services, all three types of process mining can be applied. Using the CEI infrastructure and data used by components such as the WebSphere Business Monitor, it is possible to do a wide variety of analyses including the ones shown in Figure 1.

The top-right corner in Figure 1 shows a discovered process models using the EPC notation (i.e., the process modeling language used by systems such as ARIS and SAP). The lower half shows performance related results. The bottom-left corner is a nice illustration of “extension”, i.e., the model discovered through process discovery is enriched with information about bottlenecks.

In the context of Websphere it is especially interesting to check conformance. First, using Websphere Business Modeler, a business analyst designs a service, which includes a process model and KPIs. Second, using Websphere Integration Developer, this design is implemented by an IT specialist. Third, using the Process Server, this implementation is executed. Using conformance checking, the business analyst could first check whether the implemented service actually fits (conforms to) the designed service. If not, then the KPI validation (does a KPI actually measure what the analyst thinks it is measuring?) is at stake.

4 Conclusion

The potential of applying process mining in the context of web services is huge. Data is omnipresent and issues like correlation can be addressed by using existing solutions. Moreover, the autonomous nature of services makes it interesting to observe processes as they actually take place.

Processes mining goes beyond classical monitoring components like WebSphere Business Monitor, because there is no need to model the processes beforehand. This offers several advantages. As an example, the deployment time can be reduced dramatically. Existing monitoring solutions typically require extensive modeling

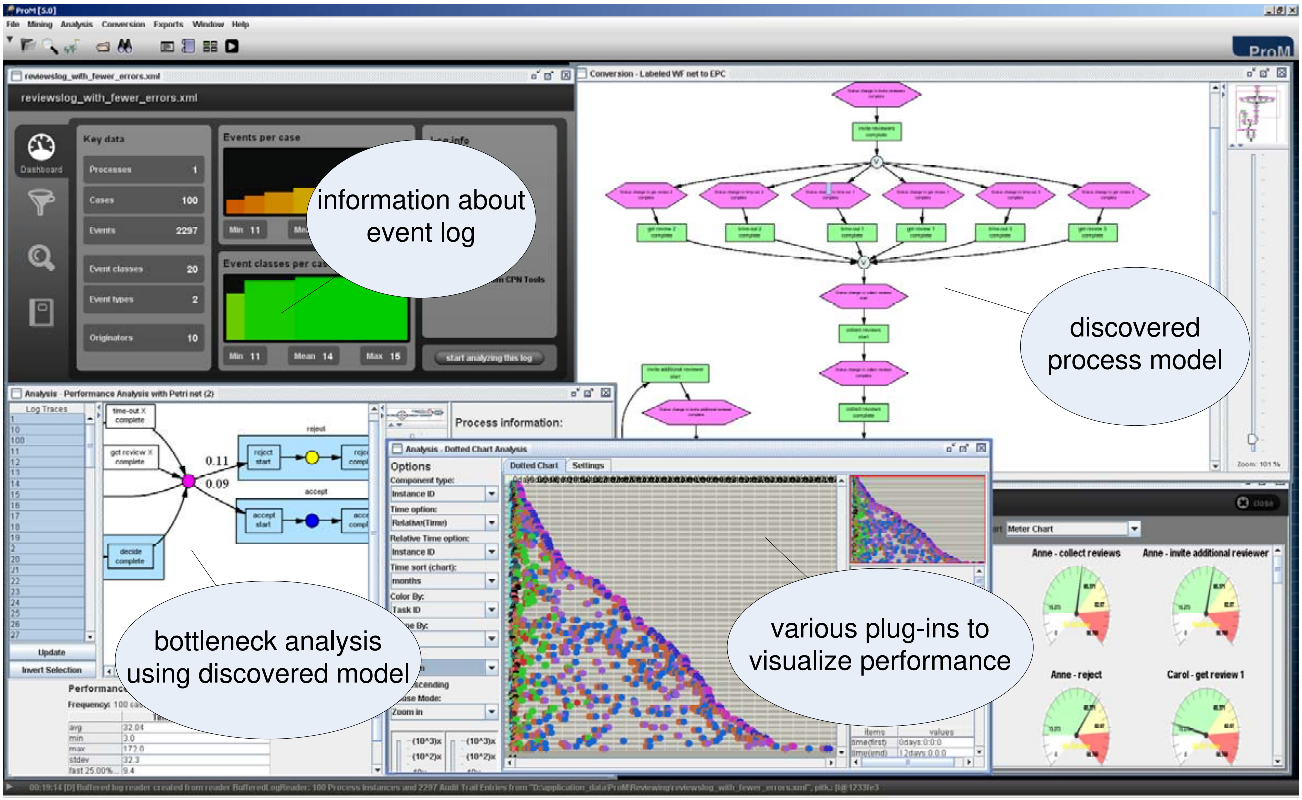


Figure 1: Screenshot of ProM showing some example results.

and configuration and cannot be changed easily. Since process mining techniques can “learn” processes, the modeling phase can be shortened and filtering techniques can be used to change the view on the process at any point in time. Process mining techniques are also able to detect (conformance) process changes and to adapt (discovery) the monitor model.

Process mining tools such as ProM have shown to be able to work with huge amounts of data and, therefore, process mining can be applied to real-life web services.

References

[1] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.

[2] IBM Corporation. Business Activity Monitoring with WebSphere Business Monitor V6.1 Redbook. `http://www.redbooks.ibm.com`, 2008.

[3] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J.F. Morar. Web Services Navigator: Visualizing the Execution of Web Services. *IBM Systems Journal*, 44(4):821–845, 2005.

[4] W. M. P. van der Aalst, at al. ProM 4.0: Comprehensive support for real process analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.

25th IEEE International Conference on Data Engineering (ICDE 2009)

29 March – 4 April, 2009 Shanghai, China

Data Engineering refers to the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The **25th International Conference on Data Engineering** provides a premier forum for sharing and exchanging research and engineering results to problems encountered in today's information society. The conference programme will include research papers on all topics related to data engineering, including but not limited to:

Approximation and uncertainty in databases	**Social information management, annotation and data curation**
Probabilistic databases	**Query processing and query optimization**
Data integration	**Database tuning, and autonomic databases**
Metadata management and semantic interoperability	**Scientific, biomedical and other advanced applications**
Data mining and knowledge discovery	**Spatial, temporal and multimedia databases**
Data privacy and security	**Transaction and workflow management**
Data streams and sensor networks	**Ubiquitous, mobile, distributed, and peer-to-peer databases**
Data warehousing, OLAP and data grids	**Web data management**
Database user interfaces and information visualization	**XML data management**
Personalized databases	**Database architectures**

Accepted contributions at ICDE 2009 will make efforts (1) to expose practitioners to the most recent research results, tools, and practices that can contribute to their everyday practical problems and to provide them with an early opportunity to evaluate them; (2) to raise awareness in the research community of the difficult data & information engineering problems that arise in practice; (3) to promote the exchange of data & information engineering technologies and experiences among researchers and practitioners; and (4) to identify new issues and directions for future research and development in data & information engineering.

AWARDS

An award will be given to the best paper submitted to the conference. A separate award will be given to the best student paper. Papers eligible for this award must have a (graduate or undergraduate) student listed as the first and contact author, and the majority of the authors must be students.

INDUSTRIAL PROGRAM

ICDE 2009 will include an industrial track covering innovative commercial implementations or applications of database or information management technology, and experience in applying recent research advances to practical situations. Papers will describe innovative implementations, new approaches to fundamental challenges (such as very large scale or semantic complexity), novel features in information management products, or major technical improvements to the state-of-the-practice.

PANELS

Conference panels will address new, exciting, and controversial issues, being provocative, informative, and entertaining.

DEMONSTRATIONS

Presented research prototype demonstrations will focus on developments in the area of data and knowledge engineering, showing new technological advances in applying database systems or innovative data management/processing techniques.

TUTORIALS

ICDE 2009 will host tutorials, relevant to the conference topics. Tutorials can be single-session (1.5 hour) or for double-session (3 hour).

WORKSHOPS

The following workshops will be hosted by ICDE 2009:

- **DBRank: Third International Workshop on Ranking in Databases**
- **First IEEE Workshop on Information & Software as Services (WISS'09)**
- **Fourth International Workshop on Self-Managing Database Systems (SMDB 2009)**
- **Management and Mining of UNcertain Data (MOUND)**
- **Modeling, Managing, and Mining of Evolving Social Networks (M3SN)**
- **Second International Workshop on Data and Services Managementin Mobile Environments (DS2ME 2009)**

For more information, visit http://i.cs.hku.hk/icde2009/

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398