How can formalization of SOA help in finding solutions for IT systems

Zdeněk Skřivánek, Karel Richta

Dept.of Software Engineering, Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25
118 00 Prague 1, Czech Republic
{zdenek.skrivanek, karel.richta}@mff.cuni.cz

Abstract. Service Oriented Architectures (SOA) are nowadays one of the most important styles in developing new information systems. SOA is attracting a lot of attention in industry as credible tool for managing large infrastructures. These systems divided into divisions have often complex models, which can contain mistakes or are informal. There are not enough current tools for testing semantic correctness of included services. Ways in research of solving such challenges are Model Driven Development (MDD) principles. We introduce the necessity of formalization of SOA in process of developing new systems and also integrating the legacy systems. We want to describe the ideas of how to achieve machine readable specifications using software tools which can then be used to verify the correctness of using the service along the required rules and its testing. We want to open two specific areas of research that is formalization of the transfer process between business and software design models and the formalization of the methods of integrating existing services.

Keywords standardization, interoperability, design, formal methods

1. Introduction

When dealing with large complex systems it is generally recognized that we need appropriate abstractions and structuring principles. Modern enterprises need to respond effectively and quickly to opportunities in today's ever more competitive and global markets.

Service-oriented architectures (SOA) are the latest approach to deliver better understanding and improved techniques to master the complexities of the modern enterprise architectures. SOA is the main architectural concept in the field of service oriented computing. SOA differs from past attempts in several fundamental ways. First, it is language independent and makes no assumption about the underlying programming model. Second, communication is no longer based exclusively on request-respond patterns (RPC/RMI) but the emphasis is on asynchronous events and messages. Third, SOA is complex. SOA sees the development of new applications and services mainly as the integration and composition of large scale services and applications rather than as a smaller scale programming problems. These differences arise from the last two decades of solving solutions for IT systems and represent a significant step forward. Definitions of SOA are given by several international bodies/organizations, including the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C). The current SOA frameworks offer service reusability, consistency, efficiency and integration. A SOA is a set of components which can be invoked, and whose interface descriptions can be published and discovered. SOA is not only an architecture, rather it is a relationship between the service provider, broker and user. The main advantage of this approach is giving the applications a way to integrate various services available online within the context of the applications specific domain and using them as needed instead of implementing the whole solution from scratch.

The rest of article is organized as follows. In Section 2 we introduce SOA as it progressed through its own history; also we add history of its vital parts. In Section 3, we explain the most important aspect of SOA, and our main interest, the SOA service. In Section 4, we discuss how formalization came into SOA, methods and research steps on this field and related works. In Section 5, we will try to define our point of view on chosen issue, our future work and we will summarize all we wanted to explain in this article.

2. SOA progress through time

SOA emerges from previous successful solutions of developing IT systems and SOA learned & followed the impact of styles such as Modular programming, Model-based development, Software components and Object Orientation methods. SOA also adapts well known technologies as Internet WWW, Open Systems, Net-Centricity, System-of-Systems Engineering and Open Distributed Processing.

From these technologies became integral parts of SOA:

- Web Service Infrastructure
- Message-Oriented Middleware
- Enterprise Service Bus
- Enterprise Application Integration

When taken from the development view SOA adapts:

- Modular programming
- Model-based development
- Software components
- Object Orientation

As times went, business pushed onwards to the software vendors so SOA have to adopt also:

- Loosely Coupled Organization
- Long Tail (Why the Future of Business Is Selling Less of More)

- Mass Customization
- Outsourcing
- Business as a platform
- Enterprise Federation
- Power to the Edge

All these mentioned technologies meets at SOA in its own style as Web Services, Enterprise Mash-Ups, Software as a Service, Real time Enterprise and ESB & Grid.

In our project we will focus on three main aspects of SOA which are important for our future work and they are: XML (content), web services (content transmitters) and their usage in SOA.

Short history of XML: Like HTML, the Extensible Markup Language (XML) was a W3C innovation derived from the popular Standard Generalized Markup Language (SGML) that has existed since the late 60s. This widely used meta-language allowed organizations to add intelligence to raw document data. XML gained popularity during the eBusiness movement of the late 90s. Through the use of XML, developers were able to attach meaning and context to any piece of information transmitted across Internet protocols. Not only was XML used to represent data in a standardized manner, the language itself was used as the basis for a series of additional specifications. The XML Schema Definition Language (XSD) and the XSL Transformation Language (XSLT) were both authored using XML. These specifications, in fact, have become key parts of the core XML technology set. The XML data representation architecture represents the foundation layer of SOA. Within it, XML establishes the format and structure of messages traveling throughout services. XSD schemas preserve the integrity and validity of message data, and XSLT is employed to enable communication between disparate data representations through schema mapping. In other words, you cannot make a move within SOA without involving XML.

Short history of web services: In 2000, the W3C received a submission for the Simple Object Access Protocol (SOAP) specification. This specification was originally designed to unify (and in some cases replace) proprietary RPC communication. The idea was for parameter data transmitted between components to be serialized into XML, transported, and then de-serialized back into its native format. This ultimately led to the idea of creating a pure, Web-based, distributed technology number one that could leverage the concept of a standardized communications framework to bridge the enormous disparity that existed between and within organizations. This concept was called Web services. The most important part of a Web service is its public interface.

Interface is a central piece of information that assigns the service an identity and enables its invocation. Therefore, one of the first initiatives in support of Web services was the Web Service Description Language (WSDL). The W3C received the first submission of the WSDL language in 2001 and has since continued revising this specification. To further the vision of open interoperability, Web services required an Internet-friendly and XML-compliant communications format that could establish a standardized messaging framework. Although alternatives, such as XML-RPC, were considered, SOAP won out as the industry favorite and remains the foremost messaging standard for use with Web services. In support of SOAP's new role, the W3C responded by releasing newer versions of the specification to allow for both RPC-style and document-style message types.

Completing the first generation of the Web services standards family was the UDDI (Universal Description Discovery and Integration) specification. Originally developed by UDDI.org, it was submitted to OASIS, which continued its development in collaboration with UDDI.org. This specification allows the creation of standardized service description registries both within and outside of organization boundaries. UDDI provides the potential for Web services to be registered in a central location, from where they can be discovered by service requestors. Unlike WSDL and SOAP, UDDI has not yet attained industry-wide acceptance, and remains an optional extension to SOA. Custom Web services were developed to accommodate a variety of specialized business requirements, and a third-party marketplace emerged promoting various utility services for sale or lease [11].

3. SOA Services

Service oriented architecture puts, as the name itself tells, the main pressure on services. Service can implement a single business process or a set of different processes that are made available for integration with other heterogeneous services. Services can be developed using a wide range of technologies, including SOAP, REST, RPC, DCOM, CORBA and Web Services. SOA basically involves three main players: the service provider, the service broker and the service consumer see Fig. 1. The service provider designs and develops a service. The service broker makes this service available to the rest of world through public registries such as Universal Description Discovery and Integration (UDDI) for web services. The service consumer locates the entries in the public registry and binds with the service provider to invoke the web services required.

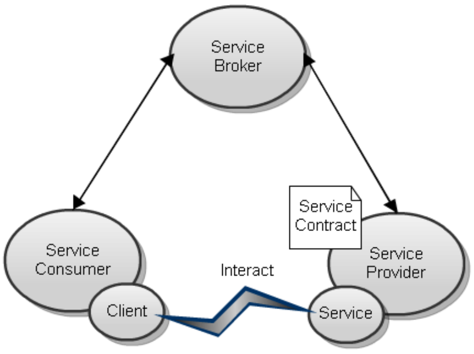

Fig. 1: Overview of requests and response flow between the actors in SOA

Services are described in a standard definition language, have a published interface, and communicate with each other requesting execution of their operations in order to collectively support a common business task or process [13]. Services in SOA are loosely coupled, supposed to be autonomous, self-contained, one have neither control nor authority over them.

Most common type of service in SOA is a web service which is a method of communication between two electronic devices over the World Wide Web. A Web service is a software function provided at a network address over the web or the cloud, it is a service that is "always on" as in the concept of utility computing. The W3C defines a "Web Service" as "a software system designed to support interoperable machine-to-machine interaction over a network".

It has an interface described in a machine-process format (specifically Web Services Description Language, known by the acronym WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. The W3C also states, "We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations."[13].

WS* is set of specifications proposed through W3C, OASIS, WS-I. It uses SOAP, WSDL, WS-Security. It is supported by IBM, Microsoft. It is designed as a technical implementation of service Oriented Architecture.

Service requests are messages formatted according to the Simple Object Access Protocol (SOAP). SOAP entails a light-weight protocol allowing RPC-like calls over Internet [13]. The SOAP request is received by a run-time service (a SOAP "listener") that accepts the SOAP message, extracts the XML message body, transforms the XML message into a native protocol, and delegates the request to the actual business process within an enterprise. SOAP is by nature a platform-neutral and vendor-neutral standard. These characteristics allow a loosely coupled relationship between requester and provider, which is important especially over the Internet where two parties may resides in different organizations or enterprises.

Requested operations of Web services are implemented using one or more Web service components. Web service components may be hosted within a Web service container providing facilities such as location, routing, service invocation and management. Web containers are similar to J2EE containers. Thread pooling allows multiple instances of a service to be attached to multiple listeners within a single container. Finally the response that the provider sends back to the client takes again the form of a SOAP envelope carrying on XML message [13]. While SOA services are visible to the service client, their Web components are transparent. The service consumer does not have to be concerned with the implementation of the service, as long as it supports the required functionality, while offering the desired quality of service.

Other technology than SOAP nowadays popular is REST. REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use [13]. A fully REST-compliant architecture is created without using SOAP at all. WSDL version 2.0 offers support for binding to all the HTTP request methods (not only GET and POST as in version 1.1) so it is closer to REST-ful web services [8]. However, support for this specification is still poor in software development kits, which often offer tools only for WSDL 1.1. Complete REST example can be found at [15].

4. Formalization

Formal methods and tools are a popular means of analyzing the correctness properties, specification of a service.

Web services are based on very minimal set of concepts: service, XML document, address and envelope. All the services must expose an interface defined using the WSDL. Several XML-based languages have been proposed for orchestration and choreography. There are many attempts to built formal frameworks for SOA managing orchestration of the services, see [2]. Services orchestration is a key issue in order to fit expectations and reach objects. Amongst the most well known orchestration languages BPEL4WS, XLANG, BPML, WSFL, WS-CDL, pi-calculus etc. The authors distinguish two layers: an abstract layer for which process algebras can be used and a concrete layer using classical services description, orchestration and choreography languages (WSDL, WS-CDL). Services are implemented with programming languages (Java, C# …).

Non-functional properties which include scalability, service reliability, and service flexibility can be assured by Quality of Service (QoS) methods. QoS is the set of techniques to manage network resources. The goal of QoS is to provide guarantees on the ability of a network to deliver predictable results. Elements of network performance within the scope of QoS often include availability (uptime), bandwidth (throughput), latency (delay), and error rate.

Description of service capabilities as automation of composition is addressed by the usage of XML-based standards for a machine readable message and interface description (i.e., WSDL). Also, orchestration languages provide the possibility of defining business processes. Besides these there are some more advised techniques we should also consider as distributed problem solving (DPS).

Each service can be characterized syntactically by its type of input and its type of output messages, i.e., its syntactic interface. The behavior of a service is characterized by the relation of input- and output messages [2]. The service perspective is the most abstract perspective within the SOA framework. The structure within this perspective defines which services are provided at the interface (black-box-view).

There are also commercial successful approaches as The Windows Communication Foundation (or WCF), previously known as "Indigo", is a runtime and a set of APIs (application programming interfaces) in the .NET Framework for building connected, service-oriented applications or IBM WebSphere Service Registry and Repository (WSRR) which is a service registry for use in a Service-oriented architecture etc.

5. Service Model

The service model tells "how the service works"; that is, it describes what happens when the service is carried out [1]. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways:

(1) to perform a more in-depth analysis of whether the service meets its needs;
(2) to compose service descriptions from multiple services to perform a specific task;
(3) during the course of the service enactment, to coordinate the activities of the different participants; and
(4) to monitor the execution of the service.

The process model identifies three types of processes: atomic, simple, and composite. Each of these is described below.

The *atomic processes* are directly invocable (by passing them the appropriate messages). Atomic processes have no subprocesses, and execute in a single step, from the perspective of the service requester. That is, they take an input message, execute, and then return their output message – and the service requester has no visibility into the service's execution. For each atomic process, there must be provided a grounding that enables a service requester to construct these messages.

Simple processes are not invocable and are not associated with a grounding, but, like atomic processes, they are conceived of as having single-step executions. Simple processes are used as elements of abstraction; a simple process may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process (for purposes of planning and reasoning). In the former case, the simple process is realized by the atomic process; in the latter case, the simple process expands to the composite process.

Composite processes are decomposable into other (non-composite or composite) processes; their decomposition can be specified by using regular control constructs such as "Sequence" and "If-Then-Else". Such a decomposition normally shows, among other things, how the various inputs of the process are accepted by particular subprocesses, and how its various outputs are returned by particular subprocesses.

6. Related works

A variety of description techniques and formalism already exists, which differ in many aspects such as separation between control, communication, structuring, formal foundation, process composition, concepts and so on. The concept of processes is introduced in Petri nets or in activity diagrams of the Unified Modeling Language. Formalization of the activity diagram semantics is possible in terms of existing formalism such as (Colored) Petri nets or by introducing new formalism. Another Petri net-based approach focusing on the control of flow aspect is YAWL. BPEL is a dominant language for the definition and execution of business process using Web services. Approaches using process algebra like ACP, CCS, CSP and variants in order to formalize work flows.

A wide variety of formal models exists for service-oriented computing. Two distinguished approaches of formalization are presented: process calculus models for expressing and analyzing service based-systems, or models for giving a formal semantic for a standard orchestration language, like BPEL [2]. Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS [2] standard executable language for specifying actions within business processes with web services. Processes in BPEL export and import information by using web service interfaces exclusively. There are three main interactions in web service composition, they are: invoke, send, and receive. In the colored Petri nets they are modeled as transitions [4].

There is need to understand and justify the role of formal engineering methods in developing services for SOA. Address the barriers to deploying formal engineering methods in business. Achieve deployment of formal engineering methods. We want to become inspired from other researchers solutions and we will bring own added value to it.

7. Conclusion and future work

It is clear that there are not enough current known tools for testing semantic correctness of included services. Our aim is to fill this gap with our own solution and programmed tools to use.

Model driven development principles try to achieve machine readable specifications and define software tools which can then be used to verify the correctness of using the services according to the required rules and its testing. The formalization of the transfer process between business models and software design models, and the formalization of the methods of integrating existing services.

There is a gap of abstraction between the formal model and concrete implementation. We should make it as small as possible. First we will seek for a service model. Patterns as components of software development could be used in model driven development or in domain specific modeling (DSM).

Available approaches do not relate available techniques to a basic, comprehensive semantic model. In order to establish an engineering approach for SOA, such a theoretical foundation of the basic concept is needed. Once we establish understanding of concepts, we can start the formalization. Operations can follow as simulation, verification, methodological support, tools etc. and we try to map them to the existing methodologies, tools, and framework if needed.

SOA has various levels of abstraction – similar to object orientation where we distinguish OO-analysis, OO-design and OO-programming. These levels address different aspects like business process modeling, system architecture and implementation. The different levels of abstraction should not exist independently but should be related to each other.

We will try to find out why some projects using formal methods but others not. Identify positive & negative experiences, opportunities & obstacles, what the added value is. And solve all our efforts to the point when integration will be possible to accomplish by non-experts.

Validation and verification must take place in order to ensure the correctness of the solution with the initial business requirements and the defined semantics. Verification and validation is possible only if concepts are clearly defined, their exact relations can be developed. Model of a service is needed. We want to describe SOA scenarios while there can be benefits from the advantages offered by formal methods.

At the business level, we do not want to consider platform-specific aspects but concentrate on core functionalities. Using components and connectors, communicating through dedicated ports only. Define vocabulary of elements as message, channel, semantics, specification, composition etc. Define and constraint relationships, communication mechanisms, and reconfiguration mechanisms. The use of UML and UML profiles as concrete notation for the presented SOA models.

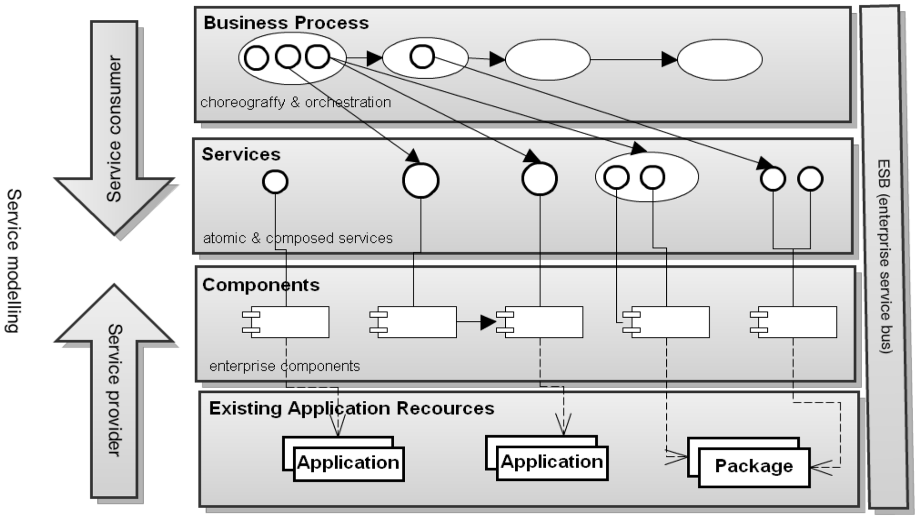

Fig. 2: Overview of service modeling, services and they relationship

Added topic one of the nowadays challenges in SOA could be the gap between transformation of business models with no component specification into a software model. All these steps we will try to finish with dedicated tools to verify of using that service.

This all is meant as verifying the process when creating new services across the needs of business, also one added interest for us will be integration of existing services into SOA by MDI (Model Driven Integration) see Fig. 2. This process of developing services across MDD/MDI we will enchant by adding component such type as a knowledge base based on CBR mechanism (Case Based Reasoning). We argue for the use of the formalization as a basis for the development of tool-supported engineering approach.

References

1. Agarwal, S.: Formal Description of Web Services for Expressive Matchmaking. Ph.D. thesis, University of Karlsruhe, 2007.
2. Allam, D.: A Unified Formal Model for Service Oriented Architecture to Enforce Security Contracts, In: *AOSD*, 2012.
3. Alonso, G.: Challenges and Opportunities for Formal Specifications in Service Oriented Architectures, Springer-Verlag, 2008.
4. Bhakti, M.A.C. – Abdullah, A.B.: Formal Modelling of an Autonomic Service Oriented Architecture. In: *International Conference of Telecommunication Technology and Applications*, 2011.
5. Bocchi, L. – Ciancarini, P.: On the Impact of Formal Methods in the SOA. *ScienceDirect*, 2006.
6. Broy, M. – Leuxner, Ch. – Fernández, D.M. – Heinemann, L. – Spanfelner, B. – Mai, W. – Schlör, R: Towards a Formal Engineering Approach for SOA. *Technical Report*, Technische Universität München, December 2010.
7. Complex Rest example:
URL: http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
8. Erl, T.: Service-Oriented Architecture a field guide to Integrating XML and Web services, ISBN 0-13-142898-5, 2009.
9. Erl, T.: SOA Principles of Service Design, ISBN 0-13-234482-3, 2008.
10. Erl, T.: SOA Design Patters, ISBN 0-13-613516-1, 2009.
11. Erl, T.: SOA Kompletní průvodce, ISBN 978-80-251-1886-3, 2009.
12. Khosravi, A. – Modiri, N.: Service Oriented Architecture Essentiality as a Best-Practice for the Development of Large Software Projects. *Journal of Automation and Control Engineering*, 2012.
13. Parazouglu, M.P. - Van den Heuven, W.J.: Service oriented architectures: approaches, technologies and research issues, The VLDB Journal, Volume 16 Issue 3, July 2007 Pages 389 – 415, 2007.
14. Rodriguez, A.: RESTful Web services: The basics, IBM, 2008.
15. Singh, H. – Singh, R.: On Formal Models and Deriving Metrics for Service-Oriented Architecture. *Journal of Software*, Vol. 5,No. 8, 2010.
16. Šelmeci, R. - Rozinajová, V.: One approach to partial formalization of SOA design patterns using production rules. In: *Proceedings of the Federated Conference on Computer Science and Information Systems*, ISBN 978-83-60810-51-4, pp. 1381–1384, 2012.
17. Verjus, H. - Pourraz, F.: A formal framework for building, checking and evolving service oriented architectures, LISTIC – Language and Software Evolution group. In: *ECOWS '07 Proceedings of the Fifth European Conference on Web Services*, pp. 245-254, 2007.
18. Wolff, T.: Using models to design business processes and services, IBM Corporation, 2011.