Hrushikesha Mohanty
Prasant Kumar Pattnaik *Editors*

Webservices

Theory and Practice

Springer

Webservices

Hrushikesha Mohanty · Prasant Kumar Pattnaik
Editors

Webservices

Theory and Practice

Springer

Editors
Hrushikesha Mohanty
School of Computer and Information Sciences
University of Hyderabad
Hyderabad, Telangana, India

and

Kalinga Institute of Industrial Technology
Deemed to be University
Bhubaneswar, Odisha, India

Prasant Kumar Pattnaik
School of Computer Engineering
KIIT University
Bhubaneswar, Odisha, India

ISBN 978-981-13-3223-4 ISBN 978-981-13-3224-1 (eBook)
https://doi.org/10.1007/978-981-13-3224-1

Library of Congress Control Number: 2018961210

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Webservice provides an Internet-based platform for conducting e-commerce. As the usage of the Internet increases resulting in smart cities and smart villages, the use of the Internet on daily basis will gradually increase manifold. Easy connectivity among people that the Internet promises has now revolutionized all aspects of human life, including business domain, particularly service sectors. Making ease for both service providers and consumers with the help of Internet-based technology has been now a primary focus. Webservice is that kind of technology currently with increasing usage. The technology just being out of academic laboratories and taking shape in the industrial world is still evolving and also getting matured for fruitful and resilient usage in the business world. Keeping the recent trends in view, the proposed book intends to cover the results that emanate from the researchers' laboratories and the applications that are being developed by software houses.

Webservice a software, being available on the Internet, provides a service being invoked by a service consumer for a purpose that is met by the service. Thus, a webservice is meant for a purpose and its developer makes the service available on the Internet so that intending service consumers can locate and interact to avail the service. In order to effectively realize such services, there is a need of Internet-based infrastructure that facilitates provisioning of webservice-based services. This architecture is generally called service-oriented architecture (SOA). Making this architecture useful, efficient, and resilient holds the success of webservices in practice. The proposed book, keeping this as an objective, charts its coverage and further illustrates the success of the technology by including experiences of professionals from the software industry.

The issues involved in provisioning webservices include webservice engineering including service choreography and orchestration. A service provider on the development of a service publishes the same at a service repository, a component of SOA that contains a minimal service description for making service consumers aware of the service. This brings out issues like service description and search for a service. The proposed book takes up these issues with fundamental contributions made by the chapter authors. In addition, they also discuss all the important techniques proposed by the other researchers.

A service required by a user may not be met by a service provider. In this case, a collaborative set of services can provide the required service. This process provisioning a service is called service composition. An engineering practice is required to adjudge collaborating services from the services available in a service repository. Both service search and composition primarily depend on the way the services are specified. So, at length, this book deals with service modeling practices. A chapter surveys finite-state machine (FSM)-, Petri net-, UML-, and SMap-based service modeling. These models are used for engineering of webservices. An engineering approach for the development of webservices, i.e., webservice engineering using FSM, is demonstrated.

Engineering is inherently associated with correctness determination. A product like webservice must ensure correctness in functioning for the delight as well as the trust of customers, and that is what a business world always looks for. So, webservice engineering has to include a study on service verification. This book takes up the issue in detail in a chapter. It discusses some important techniques researchers have proposed.

Thus, modeling a webservice using any one of the modeling specification techniques, then verifying the specified model, and generating an implementation from the model for the webservice, particularly in "Business Process Execution Language" (BPEL), are the highlights of this book. A chapter reviews the various approaches proposed for webservice composition, emphasizing input/output parameter-based approaches. Popular algorithms and their implementations are also discussed in the chapter. The role of non-functional properties in the selection of a service is a practical issue that is taken up in a chapter. First, non-functional properties are categorized, and then, the specification and discovery of these properties are addressed. Toward this, specification and discovery in a single service as well as in compositions are discussed. Both static and dynamic compositions are dealt in the chapter.

Service composition is essentially guided by dependencies of services and user requirements. In a sense it's a planning problem in finding out appropriate services by matching of one's inputs to other's outputs such that the composed service meets a user's requirements for given inputs. A chapter surveys on service composition techniques including input/output parameter-, graph-, RDBMS- and Object relational-based service composition.

Resiliency in webservices is also a sought-after feature because a service being interrupted need not be restarted from the beginning rather should be resumed from the point where it stopped. Unlike service discovery and composition, till now resiliency in webservices is a less-researched issue. In this book, a chapter is earmarked for checkpoint-based service recovery. Finding locations for putting checkpoints is a problem addressed in this chapter. Several strategies due to service specification and design analysis provide a lead to a decision on checkpoints, keeping checkpointing overhead into consideration. Further, the issue of dynamic revision of checkpoints during service execution is also investigated with the help of the hidden Markov model. Thus, the chapter addresses a new vista of research in service recovery.

Further, the proposed book contains issues on webservice security considering the priority of the issue for practical uses. Though a webservice does not specify any internal details of its operational activities, its access over the Internet makes it vulnerable to various security attacks. A chapter articulates various security threats associated with webservices in general and the countermeasures that address each of these threats along with a few case studies.

This book also addresses on webservice development cycle that emphasizes the need of webservice software development process.

Hyderabad, India Hrushikesha Mohanty
Bhubaneswar, India Prasant Kumar Pattnaik

Acknowledgements

The genesis of this book is the industry symposium on "Webservices: Theory and Practice", held as a satellite event of International Conference on Distributed Computing and Internet Technology (ICDCIT, Kalinga Institute of Industrial Technology (KIIT)) during January 13–16, 2017. The speakers at the symposium later submitted full chapters, and then, the chapters have gone through a review process for inclusion here in this book. It is always a difficult task to get full chapters from professionals amidst their busy schedules. It is heartening to note that chapter authors agreed to comply with a timeline and responded scrupulously to the suggested revisions. We extend our sincere thanks to these authors. We also express sincere thanks to Springer for giving us extra time for editing this book, else probably we would have missed editing this book.

We are thankful to ICDCIT team for its confidence on us in shaping up this industry symposium. Shri. Darpendra Narayan Dwivedy has been a great mover to ICDCIT series since its beginning. We admire his patient hearing to the demanding situations of the conference series. Founder of KIIT and Kalinga Institute of Social Sciences (KISS), Prof. Achyut Samanta is the patron of ICDCIT series from its inception. His unequivocal support has been inspiring to explore new academic vista for ICDCIT series of conferences. Both the editors express their sincere thanks to him.

The first author expresses his heartfelt thanks to School of Computer and Information Sciences, University of Hyderabad, where the major part of this book editing was carried out.

Hrushikesha Mohanty
Prasant Kumar Pattnaik

About This Book

This book intends to provide a consolidated research material for researchers and professionals working in webservice. Chapter authors are hailing from both academic and software industries. The issues on webservice functional and non-functional requirements' specification, composition, verification, fault tolerance, security, and software engineering model are discussed in this book. The first five chapters are contributed by academia actively engaged in research on webservice. The last two chapters are contributed by practitioners having industrial experience in developing webservice systems.

The contents are diligently worked with the rigor of academia and necessities of practitioners. We believe this book will be useful to the researchers as well as to the students. It can also be used as a reference book by teachers wishing to give an elective to undergraduate students. With lucidity and easiness, this book will also be useful to the practitioners who wish to know the theoretical grounding on the subject.

Contents

About the Editors

Hrushikesha Mohanty possesses a professor position at School of Computer and Information Sciences, University of Hyderabad, Hyderabad, and now on lien to work at KIIT, Bhubaneswar, Deemed to be University. He has received his Ph.D. from IIT Kharagpur to start his professional career at Electronics Corporation of India Limited (ECIL), Hyderabad, and then to join in academic at University of Hyderabad. At ECIL, he took part in developing an indigenous real-time system for the Indian Air Force. His research interests include distributed computing, software engineering, and computational social science. He also keeps a keen interest in the literature and regularly writes in literary Odia magazines, newspaper, and his blog MoKatha.

Prasant Kumar Pattnaik has received his Ph.D. in computer science, is Fellow IETE, is Senior Member IEEE and is Professor at the School of Computer Engineering, KIIT University, Bhubaneswar. He has more than a decade of teaching and research experience. He has published more than 50 numbers of research papers in Scopus-indexed international journals and conferences. He also co-authored and co-edited nine books. His areas of interest are mobile computing, brain–computer interface, and cloud computing.

Service Modelling and Verification: A Formal Approach

Deepak Chenthati and Hrushikesha Mohanty

Abstract Increasing number of users on web has attracted a large number of businesses to be made available on web. The growing demand for webservices requires a systematic approach in service development. For the purpose, this chapter reviews various models for service specification, composition, deployment and monitoring. Particularly, the chapter reviews some well-known models like UML, Petrinet and state machine used for service modelling and verification.

1 Introduction

Web has become a common platform for different enterprises with increasing availability of the Internet. Also with standardisation of SOA technologies, enterprises tend to expose their business as webservices. Enterprises to meet the market needs would resort to the methods that enable them to build a service quickly and effectively without errors. In this process, either the existing services are used for building a service with enhanced/higher requirement or services are generated from models. Here, initially, the services are modelled using formal methods, viz. UML, Petrinet, state machine, etc. A modelled service is used as a source/reference for automatic generation of executable codes. The advantage of service modelling is in three folds. Firstly, it gives a visual representation of a service that is being modelled giving service provider a scope to analyse its flow of execution. Secondly, the errors due to manual coding are eliminated. Thirdly, the deployed/running service can be verified

D. Chenthati (✉)
Teradata (R&D) India Pvt Ltd, Hyderabad, India
e-mail: chvcdeepak@gmail.com

H. Mohanty
Kalinga Institute of Industrial Technology, Deemed to be University, Bhubaneswar 751024, India
e-mail: hmcs_hcu@yahoo.com

H. Mohanty
School of Computer & Information Sciences (On leave), University of Hyderabad, Hyderabad, India

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_1

against the model either for testing or for checking the abnormal behaviour of a service. These advantages have motivated several research works to address modelling of services considering different aspects.

A webservice has a business logic available on web and it is exposed to the external world as an interface. The major concerns during modelling of services would be (i) to hide the business logic, (ii) to generate an interface as per a specification standard, i.e. WSDL, (iii) composing a service when a single service does not meet a need, (iv) monitoring a deployed service to capture quality details, (v) security aspects of a service and (vi) provision for model-based verification. Each model follows different rules for modelling and code generation considering some or all of the aspects stated above. These rules in turn form the guiding principles for verification of webservices.

The concern of business logic is addressed with orchestration of a service where the activities of a service are in order for execution. Composition of services involves issues with matching of interfaces and protocol for proper communication among constituting services. Matching of I/O interfaces would initially find a set of services that are further checked for protocol match. Choreography of webservices addresses this issue assuring observance of an agreed protocol among chosen services. Further, the verification of a service specified with rigour of models addresses issues related to structural and behavioural correctness. However, here no claim is made on completeness of model-based webservice specification.

Section 2 of this chapter gives a detailed review of some models used for service modelling. The next section takes up issues on verification of services. Modelling of services is discussed with respect to approaches based on UML, Petrinet and state machine. Each of these approaches is discussed with different enhancements and proposed standards. Finally, the chapter concludes with a brief concluding remark.

2 Modelling of Webservices

2.1 UML-Based Approach

Unified modelling language (UML) has been successful in modelling software systems that is reflected from its popularity in software industry. Webservices being a contemporary technology-driven means to deliver services, needs well-thought design principles for success. UML being a proven modelling approach is considered for the purpose of webservice design. This, in fact, brings both academia as well as industry professionals to the same platform to contribute together in development of UML-based methodology for webservice development. In this section, we will survey some works in highlighting UML-based webservice design approach.

Model-driven architecture (MDA) is a popular method to develop software systems. UML has a mechanism to model both structural as well as behavioural specifications of a system. In case of webservices, UDDI that maintains service locations also comes to picture. Further, composition of services is to be modelled. Service compo-

sition modelling specifies a way messages between two services are to be exchanged. Compatibility of message passing protocol specification leads to successful service composition. UML class diagram models interface while actions are modelled by UML behavioural diagrams like use case, activity and interaction diagrams. The aspects particularly to be modelled with respect to webservices include service artefacts, interfaces, data access, execution and communication error handling, execution tracing, usages of ontology, designing service communication patterns, specifying service locations and service metamodelling. A brief survey on usability of UML in specifying webservice composition is given in [1].

W3C, the organisation engaged in developing standard has proposed a specification web services choreography description language (WS-CDL) for specification of webservices. Service providers specify services and for a consumer request, services are searched and an appropriate service is selected for providing service. Researcher in [2] proposes a variant of UML-2.0 called UML-S to specify a system to be developed with WS-CDL specification. This requires a one-to-one mapping between the two specification approaches, i.e. WS-CDL and UML-S. Service modules specified in WS-CDL are specified in UML component diagrams. A sequence of actions a module performs is specified by UML activity diagrams. Behaviours due to each role specified in WS-CDL is modelled by UML state diagrams, whereas descriptions for each state is modelled by class diagrams. With an experimentation, authors have shown the use of UML-S-based specification in modelling and verifying a service specification.

In [3], researchers have proposed another extension to UML called WS-UML for webservice design. WS-UML is alike UML, proposes graphical annotations for specifying webservice concepts like service security, composition, location and trace of execution. This work stands out among similar works by allowing to specify service location. It also allows user service providers as well as consumers to trace service execution. A composed webservice is to provide an integrated service to a user in accordance with its requirements. This requires selection of services and composition of services. This process requires certain essential aspects that a service designer must address. WS-UML provides that mechanism for specification.

First, question of service selection comes. Selection of a service not only is based on service functionality say by specifying input and output but also its quality of services. Quality of a service for a webservice is specified by service cost, reliability and execution time required for service delivery. WS-UML provides means to annotate a service by these information in a given syntax. Further, on selecting services, compositions are done. In the process, first composability criteria are to be evaluated. Each service specifies its conditionality for composition, e.g. protocol requirements for passing messages. Other than this communication, there is a need to specify security aspect. Designer makes a provision for defining security mechanism for both service provider as well as consumer. Hence, during invocation of service security aspects for both provider and consumer can be ascertained. Location where a service is hosted on the Internet is to be specified for easy access of users. This access is to be of course automated. For the purpose location specification is necessary. This provision is made in WS-UML. Next is execution tracing that is essential in business

world. Once a service is invoked, it is required to trace the execution to check whether the service provision is being done as per requirement, i.e. service-level agreement.

WS-UML provides means to specify the desired states by which service execution must pass through. This design provision helps in service error finding and thus service maintenance.

Later a work [4] proposes a process of webservice design using UML. Webservice is seen as a choreography of several loosely coupled services. They collaborate to solve a common task regardless of their programming languages and environment. In order to model such collaborative system, UML is considered as a natural choice. Because, component diagram, sequence diagram and state diagram can model service components, their functionalities, interactions and state changes during execution. The paper proposes seven steps for webservice choreography modelling and verification. First, WS-CDL structural properties are modelled by component diagrams. In the second step, sequence diagrams used to model component interactions are translated to state diagrams. Because, state diagrams can be used for model verification. Then, in the third step, the abstract data model for WSDL is modelled by class diagrams. State diagrams are now enriched by class diagrams with tags stating the changes occurring to data in different states. Then, in the fifth step, enriched state machine diagram is translated into language of a model checker like SMV. Then, in the following sixth step, informal requirements, if any, are specified by designers for verification. At the last step, a model checker is used to verify a model behaviour. This work shows not only how to model a webservice but also to verify its design specification.

In another work [5], use of RT-UML to model and verify orchestration of webservices is reported. The main purpose is to verify time aspects of orchestration. For this purpose, RT-UML is considered. Researchers have found one-to-one mapping between RT-UML constructs and WS-BPEL. A top-down design approach is proposed for developing webservices. During analysis, time constraints that webservice orchestration must satisfy are found. And these time constraints are annotated to sequence diagram. Then, model checking approach is used to discover specification errors. This is done by translating RT-UML diagrams into timed automata that are used to perform model checking. Verified RT-UML design diagrams are then translated to WS-BPEL. The elements used by RT-UML for webservice design includes RTDelay, RTEvent, RTAction, RTreset and RTclock. The elements have their semantics that are well understood by their names. These elements have corresponding translations to WS-BPEL language. Thus, the work provides a systematic approach in design of webservices using RT-UML particularly keeping time constraints in view. The work is in importance for bringing time aspects in modelling service orchestration.

Recent work focusses beyond UML while modelling webservices. This is so for variety of emerging distributed architectures like cloud and mobile systems. The argument on utility of UML is raised for heterogeneity these platforms bring in. It cites the inability of UML diagrams in representing all the heterogeneities as inherently class diagrams model homogeneity of design entities. As a solution to this problem, researchers in [6] have proposed domain specific language (DSL) to design

webservices that runs in different environments. Design abstraction that is common to all the implementations is extracted and then the abstraction extended for each implementation. This design extension is carried out to meet the requirement of a platform. The researchers have proposed simple web service modelling (SWSM) domain specific language. Webservices in multitenant platforms are specified in SWSM language. Then, it is customised for a specific tenant. The challenge in having such a language is to specify webservice architecture at an abstract level separating logic from its technical implementation aspects.

SWSM language proposes a syntax that elegantly specifies an abstraction of webservices. The terms used in syntax are *Webservice*, *Port*, *Binding*, *Datatype*, *Operation*, *Message* and *OperationBinding*. Modelling of webservices are carried forward representing principal elements using these terms. The design approach is essentially top-down. The steps in developing webservices include modelling using SWSM language, enhancement and automatic validation of webservice models, code generation and code refinement, and refactoring and testing. Model-driven development of software systems is successful and UML has given impetus for it. But, non-UML-based modelling approach is also followed for its conceptual clarity and simplicity. This work is of that kind introducing non-UML-based webservice modelling. Next sections of the chapter will review some of these well-known modelling schemes based on finite state machine (FSM) and Petrinet.

Table 1 gives a summary on research works mapping webservice specifications to concepts in UML and this table is cited from research work in [3]. WSDL is generated from UML profile [7], an extension to UML class diagram with service-related concepts, viz. Service, Port, Port Type. Jeng and Tsai [8] look services as components and proposed UML profile based on service component architecture (SCA). Jeng and Tsai [8] views from security angle and has proposed extra functional properties to record service invocation logs, control access to operations and encryption. Composition of services is focused on [9].

2.2 Petrinet-Based Approach

Webservice while taking business on the Internet to a reality there has been spurt in research activities in this field. Service modelling and composition are the two important issues under consideration, while modelling concentrates on both static as well as dynamic aspects of webservices there has been further interest in modelling temporal as well as asynchrony aspects of webservice execution. For the purpose, Petrinet is a chosen model. Many have tried with variant of Petrinet models and shown their utilities in webservice modelling, code generation and more importantly in design verification. Here, we will review few papers that only represent types of work researchers engaged in this field. We do not claim the review is complete or exemplary. For the purpose, here, we believe the papers selected here for review are representative.

Table 1 Review of webservice specifications in UML

View	Concept	References
WSDL	Class UML stereotyped "WebService"	[7, 10]
	Class UML stereotyped "Port"	
	Class UML stereotyped "PortType"	
	Tagged value relative to a service and the port {URI="/ "}	
SCA	Component stereotyped "ServiceComponent"	[11]
	Tagged value relative to an interface {R.uri= "Operationuri"}	
	Class stereotyped "ServiceInterface"	
Security	Note stereotyped "Login"	[11]
	Note stereotyped "Log"	
	Note stereotyped "Encryption"	
Composition and orchestration	Class stereotyped "ServiceWebAtomic"	[9, 10, 12]
	Class stereotyped "ServiceWebComposite"	
	Class stereotyped "ModeledOrchestration"	
	Activity stereotyped "ImmediateStep"	
	Note stereotyped "DataTransformation"	
	Activity stereotyped "DataMapper"	
	Tagged value relative to the activity "ImmediateStep" {DomainObject=}	
QoS	Class stereotyped "ServiceQuality"	[10, 12]
Community and function	Class stereotyped "Community" Class stereotyped "Function"	[10, 12]

A webservice at higher level can be viewed as a collection of interacting modules. Architecture of a webservice is built with these modules. Configuration of modules, the way these are connected, presents an architecture for a webservice. The paper [13] presents WS-Net, an architectural description language for specifying webservice architecture. The language is based on Petrinet semantics accompanied with object-oriented paradigm. These two aspects of WS-Net provide a mechanism to specify and verify a webservice model. It is also useful to monitor dynamic behaviour of webservices. WS-Net uses coloured Petrinet for higher level design.

WS-Net is executable architectural language. Webservice model is seen here as three-layered model consisting of interface net, interconnection net and interoperation net. Interface net models network of interfaces and their possible transitions. Each service component is treated as a place and the modules that can be invoked from the place are connected by transitions. Thus, Petrinet becomes an obvious choice for webservice modelling. The next layer below, interconnection net models the foreign transitions that invoke the components that are external to a component. The next lower level models behaviour of a component, that is, a sequence of invocations of the units that make a component. Coloured Petrinet is used for understanding like to distinguish models at different levels. It is shown that the executable architectural language becomes useful for both modelling as well as verification. Adopting object-oriented paradigm in WS-Net, understandability as well as agility in webservice design are achieved.

A new concept in modelling and analysing webservice composition using Petrinet is proposed in [14]. This paper reviews all existing methods and proposes a method that illustrates modelling issues at different levels using coloured Petrinet. Basic questions like *who* and *why* for service composition are answered in this paper. Among existing set of services which one is to be selected for composition and how does the selected one can interact with other services for a given user requirement are answered by the model. The unique issue the paper takes up is selection of a service that is to be guided not only by functional requirements but also nonfunctional requirements. The cited work extends formalism of CPN to model transaction-driven composition process. Further, while modelling quality of service (QoS) is also taken into consideration. The composition process also considers system modularity. The process is standardised to make it compatible with Web 3.0. This modelling helps service composition operationally simple. A service user just puts a query stating its requirements and then the composition process matches the Petrinet-based modelled webservices and finds the services that meet both functional and nonfunctional requirements.

The paper [15] has proposed a high-level Petrinet for service modelling. The modelling concept is based on G-Net [16]. G-Net provides an algebraic specification to specify modular complex systems. It has two levels of abstractions; one is to specify inner working of a module and other on modular interaction. G-Net follows principles of object-oriented design implementing encapsulation of a module that shields a module from external interference. It also provides a mechanism for sharing resources among modules, i.e. by G-Net abstraction. For this mechanism, G-Net is

found suitable for module-based complex system design. This paper [15] extends G-Net to make it suitable to specify webservices.

A webservice is a tupple <*NameS; Desc; URL; CS; SGN*> where

{NameS: is the name of a service used as its unique identifier.
{Desc: summarises a service functionality.
{URL: for invocation of webservice.
{CS: a set of component service a service has.
{SGN: (GSP, IS) is the G-Net modelling the dynamic behaviour of a service.

GSP represents service abstraction specifying constituting executable methods and attributes. IS represents internal structure of a service showing a set of transitions and their sequence of occurrences. Based on this idea of G-Net, the paper proposes operators like sequence, parallel, alternative, iteration and random sequence to model service composition with collaborating service components. The paper also proposes four more operators like discriminator, selection, refinement and replace. The paper has shown the possibility of service composition using these operators. The unique point the paper demonstrates is the transformation of G-Net-based algebraic specification of a webservice composition to a Petrinet model, albeit it is complex but executable as well as verifiable. Table 2 gives an overview of the existing research work and how Petrinets are enhanced to model service compositions [14].

2.3 State-Machine-Based Approach

State-machine-based software design is not new. There is a natural correspondence between system and finite state machine. A software system passed through several states, that is, represented by states in a finite state machine. System execution is considered as a dynamic behaviour of a finite state machine. A work [48] demonstrates possibility of modelling complex systems with a set of finite state of machines. Each machine represents a module. The states of a module represent a node in its corresponding state machine. A system execution is modelled by interactions among modules; an interaction is alike to a function call. It models repetitive as well as recursive calls to modules. Execution trace is a path that runs from a start state to end state of a finite state machine. It presents a hierarchical representation of state transitions to model behaviour of a complex system. Modularisation of finite state machines is a concept the paper proposes. This enables to optimise finite state machine by grouping the system states that are common to different execution paths. Further, the paper claims utility of the proposed modelling process as the modelled finite state machine (FSM) that is easily transferable to code enabling automated code creation. On considering success in finite-state-based system building, there has been research in finite-state-machine-based webservice modelling. The modelling effort here includes service modelling, verification and automatic code generation. Here, we review some works to present the utility of FSM-based webservice modelling.

Table 2 Petrinets for service modelling

Approach	Objective	References
Classical Petrinet	Propose a Petrinet-based algebra to capture the semantics of WS composition and to formally model a composite WS, which is the first step to allow the verification of the composition and the detection of inconsistencies within and among WS	[17]
Coloured Petrinet	Propose a coloured Petrinet to model types of resources managed by WSs	[18]
Time-constrained Petrinet	Propose a time-constrained Petrinet to model and analyse time-constrained WS composition	[19]
Generalised associative Petrinet model (APN) (fuzzy Petrinet)	Define automatic WS selection based on manual user specifications and using fuzzy Petrinet	[20]
Adaptation of classical Petrinet and Open workflow net	Transform a BPEL process to a Petrinet in order to allow process verification	[21–23]
Open workflow net composition net	Transform two or more BPEL processes to Petrinets and compose Petrinets in order to detect WSs incompatibility	[24, 25]
Open workflow net with coloured Petrinet	Transform two or more BPEL processes to Petrinets, compose Petrinets in order to detect WSs incompatibility, and add mediator transitions to correct partial incompatibilities among WSs	[26–28]
Classical adaptation of classical Petrinet Hierarchical coloured Petrinet	Transform BPEL or WSCI processes into Petrinets in order to verify reachability, safety and deadlock	[29–32]
Time Petrinet, adaptation of classical Petrinet	Transform WSADL specifications into Petrinets to evaluate aggregated QoS criteria of the composite WS	[33–35]
Adaptation of classical Petrinet	Generate Petrinet from OWL-S definition of WS for checking the correctness of WS specifications and the replaceability of (sub)services	[36]
Time Petrinet	Define timed Petrinet representation of WSs flow from WSDL specification	[37]
Prioritised timed extension of coloured Petrinet	Generate Petrinet from WS-CDL definition of composite WS for simulating timed or prioritised interactions among component WSs	[38]
Classical Petrinet	Propose an automatic QoS-transactional WS selection based on classical Petrinets	[39]

(continued)

Table 2 (continued)

Approach	Objective	References
Classical Petrinet	Propose an automatic QoS WS selection based on Petrinet coverability	[40]
Coloured Petrinet	Propose an automatic WS selection based on coloured Petrinets	[41]
Coloured Petrinet	Propose an automatic QoS-transactional WS selection based on coloured Petrinets	[42]
Coloured Petrinet	Propose framework for reliable execution of transactional composite WS based on coloured Petrinets	[43, 44]
Compensation paired Petrinet	Propose compensation paired Petrinet to model, evaluate QoS, verify reachability and deadlock, and control the execution	[45–47]

Webservice is a piece of code that gets executed on getting a service order. Usually, a service is hosted on the Internet and a user invokes the same through an interface. A user can be a person or even could be another service. Being invoked, different blocks of a service get executed at different contexts that make states. Context changes represent transitions. At the end of a service, result is handed over to the service invoking agency through an interface. So, a service abstraction includes both service internal module and interfacing module. This paper [8] discusses webservice design framework with the help of StateJ framework. StateJ is basically a framework to model event-based applications. It has two major components; one state transition engine and the other for event notification service. State transition engine invokes state modules at different contexts (state management service (SMS)). That can be viewed modelling of intraservice communication (request management service (RMS)), whereas event notification service (ENS) models interservice communication. StateJ uses a finite state machine to model the behaviour of a service. Authors have claimed the usability of StateJ as a natural framework for service modelling. Further, the framework provides a natural flexibility in designing a service, enabling a designer to change service component on fly. A scalable design is also possible due to StateJ. A high-level implementation of an architecture for service design is shown in Fig. 1.

Service composition is a process of generating a new service from existing services. This needs to check compatibility of constituent services. This compatibility could be at two levels. One is input and output level and the other is at conversational level. For the prior case, one service calls the other on providing expected inputs and expecting the required outputs. Later case, two services proceed with conversation while on execution. In the first case for service composition, one needs input–output match, while for the later protocol match is required for finding compatible services.

The paper [49] deals with the problem of service composition. Particularly, it deals service enactment. Service enactment means finding a service execution plan that confirms service requirements and constraints as provided by service providers and

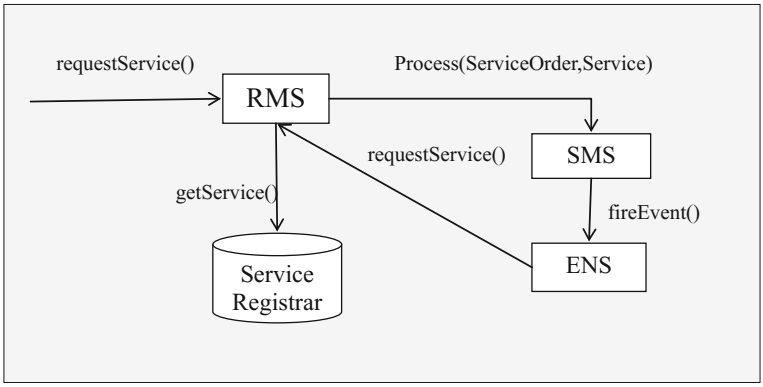


Fig. 1 High-level implementation of StateJ architecture

consumers. However, both service providers and consumers are usually not willing to expose their business details on service repository. This throws a challenge for service composition. This paper provides a mechanism that does not need stakeholders to expose their business secrets but still enables to compose a service with possible execution plan that satisfies their requirements. The proposed approach uses finite state machines to model constituent webservice operations and their interdependencies, security constraints and access control policies. It incrementally generates service enactment plan. The paper also suggests commutative encryption-based techniques to preserve privacy and security.

The paper [50] provides a formal approach for B2B collaboration among webservices. Existing webservices collaborate to provide different services that as such do not readily exist on web. In this context, study on matchmaking process takes a lead. This paper uses deterministic automata for the purpose. Before matchmaking of webservices input–output matching technique is a well-known technique. It says two services are composable when input of one matches to output of other. This standard paradigm at times may not work when there is a mismatch in their process descriptions. In spite of input–output match between two services, there could be mismatch when a service at a state needing some input from other may not get it if there is no corresponding output from the other service. This paper proposes finite-state-machine-based approach for service matchmaking. A service specification is seen as a sequence of state changes where each state is annotated with a message that has sender, message content and receiver associated with. Composition of services is viewed as intersection of state sequences services undergo during their life cycles. This is based on simple reasoning on message transmission verifying whether the right message is sent by right sender to the right receiver. In case of null intersection of state sequences of two services, service composition becomes impossible. In case of partial match, negotiation among services takes place to make the services agree to make a composed service. This idea of negotiation for composition is found new with the work.

Role of communication in service composition is further researched in [51] showing the use of Mealy machine that accepts a language meant for webservice composition. In addition to idea proposed in [50], this paper introduces a notion of global controller that keeps a watch on message passing at different webservices. In order to realise asynchrony in message passings, it introduces a notion of *prepone* operator that brings in random delay to message passing. A projection of global communication in view of a participating service presents local picture on message communication. Message passing between two webservices is formalised by *project-join* of global communication. A composite service is considered as Mealy implementation that generates conversations whose projections are consistent with participating individual webservices. However, there are some communications among Mealy machines which cannot be formalised as project-join of a regular language. Hence, the researchers propose a formalism for webservice communication that is useful for service composition. With this protocol in addition to modelling interplay between two Mealy machines, global behaviour can also be modelled. The research shows that for a given regular language it is possible to get a conversation by applying prepone and project-join closure corresponding to the set of conversations a set on Mealy machines can perform for a given language. The second result spells the conditions at which a set of conversation from a given language applying prepone and project-join closure is achievable.

Modelling and verification go hand in hand. On reviewing some of the prominent modelling techniques next we will review some important works on webservice verification and while doing so, we may touch upon some modelling issues. This may lead to some repetitions as well as additions in the next section. Readers are foretold of the fact before we proceed for review on webservice verification.

3 Webservice Verification

Software verification is a crucial activity for achieving fault-free product. There has been active research on webservice verification. State-machine-based verification is one of the techniques pursued in [52]. Two issues, i.e. communication and service module verifications are considered here. Communication between two composing services is expressed in XPath expressions. This defines guard conditions on communications. Service module defined in BPEL code is tested. For testing the later, a service BPEL code is translated to state machines that models threads execution contained in BPEL code. Communications among composing services are also modelled as state transitions. Communication conditions label state transitions. Promela is a target language to translate service implementation to finite state machine. This executable language is run on SPIN to simulate behaviour of a state machine and to identify correctness of the model. The model assumes unbounded message queue while modelling message communication. In practice, webservices are engaged in bounded communication. SPIN verifies synchronisation. SPIN can only operate with bounded message communication and partial verification of synchronisation can

also be done. The paper proves for asynchronous communication if there exists a bounded number of messages that need to be passed among webservices for synchronisation and then it is possible to verify synchronisation for the same set of messages in case of synchronous messages communication. Thus, communication protocol for a service composition is verified with the help of finite-state-machine-based models.

A formal verification approach is proposed in [53]. The approach is useful to verify workings of a service that is composed of several services. The composition follows matching of conversations among participating services. Most services perform stateful conversation so the method takes this feature while verifying working of a composed service. The novelty in this approach is its global view on modelling communication among participating webservices. They have proposed coloured Petrinet model for the purpose. This model is based on the control flow patterns of BPEL4WS where each node represents a node and an arc represents message passing. In a way, CP-net models control flow among services that make a composed service. Webservice activities are seen as message passing and action invoked thereof. On modelling, a service verification is taken into consideration. The thrust is given to protocol conformance, i.e. while putting a service in a composition the issue of conversation among services is to be relooked to ensure whether conversations among participants are well defined and so also there of actions. In addition, general criteria such as liveness, boundedness and reachability are verified by applying CP-net analysis technique. A similar work is reported in [54] that models webservice developed in BPEL to coloured Petrinet net (CPN) by mapping BPEL constituents to Petrinet nodes, transitions and colour tokens. By virtue of Petrinet concurrency, communication and synchronisation properties are being modelled. Then, using CPN tool, service behaviour is analysed and verified.

In [55] a generic framework called VERBUS is proposed for verification webservices. VERBUS framework integrates several formal verification tools like Spin and SMV. A webservice can be developed in many languages. Usually, from implementation, a formal model is to be extracted and then formal model is to be verified be it a CPN or a FSM. Thus, verification process is used to be tied with implementation language. This work is distinctly different in suggesting an intermediate specification language to which any type of implementations can be translated into. A service specification written VERBUS intermediate language is compiled to produce executable for a model checker. The paper reports verification of invariants, goals, pre- and post-conditions, activity reachability analysis and temporal properties. A logic-based approach for service verification is reported in [54]. The researchers take up service composition in two steps; one identifying services required for the purpose of a service consumer and other on selecting the right one for the identified services, while the first step ensures the functionalities and the second one ensures the quality of service. Thus, the method not only ensures selections but right selections. The first one tells signature matching and the other one as specification matching. Signature matching is based on matching of input–output parameters among constituting services, while specification matching ensures satisfaction of constraints defined over input–output parameters. For specifying, it uses both temporal logic action (TLA) and first-order logic (FOL), while the former specifies service signature the later

one specifies system behaviour. For a given query, equivalent logic expression in TLA is formulated and then services satisfying the expression are identified. Further, constraints defined on services are evaluated to choose the right ones. The key for verification rests on formulating right kind of constraints that participants of constituting services would satisfy.

Unified modelling language being de facto industry standard in software development, the work in [56] takes up the issue to show modelling as well as verifying a webservice. The process goes through several stages. Specification written in web services choreography description language (WS-CDL) is first converted to component diagram and this diagram is put through static analysis to verify some structural properties like dependency and connectedness. Secondly, a sequence diagram is generated that depicts service behaviour. In third stage, sequence diagram is used to derive state diagram that is enriched with control and data flows. Then, state diagram is translated to SMV language so that SMV model checker is used to verify different design properties that are allowed with respect to application domain. The work demonstrates the verifiability of a service specified in WS-CDL transforming into multiple UML models.

The idea verifying early in the process of development is always preferable for its utility in getting a verified correct design so that implementation will be correct provided no error has crept in during translation of a design to an implementation. The paper in [57] takes up this approach. A UML specification of a webservice in message sequence charts (MSC) is converted to finite state process (FSP) notation. And then, the specification in FSP is traced for analysis. Such notation specifies design views and then these are matched with user views. In case of match, the design is considered correct. The properties they verify include process and partners, message passing, sequence, loops and concurrency. In general, the paper proposes a technique that supports modelling, verification as well as implementation.

Petrinet-based static analysis of a webservice is presented in [58]. A process language like BPEL has been adopted for webservice implementation. BPEL provides language primitives for message passing. These primitives combined with several control features including sequential, branching, parallel and synchronisation are implemented. A BPEL process can be translated to a Petrinet. And then, making use of Petrinet analysis technique, control flow of a business process can be verified. A work towards static analysis of webservices is reported in [58].

The paper [59] presents verification of webservice choreography by analysing their cooperations. It ensures correctness in their protocols. This ensures a right business partner talking to right one on a purpose. This is ensured by their protocol matching. It also ensures to check deadlock-free communication. The approach proposed includes first characterisation of each activity with a set of operational rules. These rules clearly define permissible functional behaviour of a service. Service activities include both basic communication activities like request, reply, etc. and structured activities like flow, while, switch, etc. that model business workflow. Secondly, compatibility rules are defined to adjudge the compatibility of two processes. That means, two processes can proceed their activities in cooperation by message

passing. Then, algorithms for that check are formulated. The algorithms check both progresses of a service execution as well as their communication.

Zhao et al. [60] proposes a technique for webservice verification. It also proposes a method for generating test cases from BPEL logic so that dynamic behaviour of a service can be verified. From these, some test cases are selected with respect to counterexamples of model checking output. A service in BPEL is translated to LOTOS and labelled transition system (LTS) is translated to test and testing control notation (TTCN) behaviour tree. Keeping runtime behaviour checking in mind, the process of testing puts importance in determining frequency of testing and identification of operations to be tested. More testing should be done without interrupting service execution. A tool is developed to model BPEL with LOTUS. The EVALUATOR 3.0 tool is applied to generate LTS as output. Making use of such a tree safety, liveness and fairness properties of a service are verified.

A recent work on service behavioural compatibility and similarity study using Petrinet is reported in [61]. Seamless running of a composed service depends on compatibility of constituent services. This has been verified by matching of individuals protocols. But, this paper proves that study on termination is good enough to verify correctness of a composed service. For the purpose, services are required to be well structured. It talks of resilience by replacing a service equivalent to a failing service. Researchers define conditions to study on context-independent similarity among webservices. For the purpose, a notion of behavioural compatibility is formulated. A formalism called service workflow net (SWN) based on coloured Petrinet (CPN) is developed to model webservices. Through this formalism some important runtime behaviours like message passing, buffering and choice making are anlaysed. With the help of these studies, reachability termination and proper termination are verified. Further, for finding similarity weak soundness is considered so that conditions can be relaxed to find suitable compatible alternative matching services to replace faulty ones. A tool supporting the proposed theory for verification and similar service detection is developed.

Keeping developments in cloud computing and deployment of services on cloud there have been a concern on pace of development of service technology. In that context, the work in [62] addresses the issue of dependable services. For modelling that service behaviours are characterised to operational behaviours and control behaviours. Runtime coordination among those behaviours is achieved by message passing. Based on this modelling, an approach for service verification is proposed. The approach is based on symbolic model checking. Verification properties are extracted from control behaviours in form of temporal logic formulas. And these formulas are verified with operational behaviours. This checking is done using NuSMV model checker. Based on this approach, a set of tools are developed for service engineering confirming the possibility of automating a set of tasks including assistance in service specification, detecting design problems, debugging and monitoring service behaviours.

Resilience in webservices has become prime interest for making it useful for business houses. This has led to understanding mistakes and provisioning to overcome such mistakes. The paper [63] takes up this issue viewing webservices as event-

based transactional systems. The transactional behaviour is studied in both design and runtime to find out design errors as well as runtime errors. The method proposed includes first to translate service transactional patterns to formal expressions using event calculus. These expressions are evaluated to check transactional consistency before design and after run. Prior conditions to a transaction expressed in event calculus are evaluated. And then posterior conditions are tested using service logs. A webservice has a life cycle consisting of states, viz. initial, active, cancelled, failed, compensated and completed. It has also a set of transitions, viz. activate(), cancel(), fail(), compensate(), complete(). Pre- and post-conditions are associated with a state change. Verifiability of a service is made possible by satisfiability of these pre- and post-conditions.

Data-centric webservices require interfacing of data repository and services as per the user requirements. Further, in case of composed services, constituent services in a sequence use data. In the process, data creation, deletion and modification take place. Correctness of data usages is required in assuring correctness in data-centric service operations. A service-contract-based strategy is proposed in [64]. For a given set of data operations, a formalism is enforced. This contains conditions associated to a data operation. This is called service contract. For a given set of data, an interfacing formalism is defined so that when a service interfaces with a repository for a data operation the contract is validated. Similarly, in case of a service, service interaction contracts for data provider and consumer are defined. These contracts are to be satisfied during execution of a composite webservice. The paper presents an implementation of the proposed approach, Java modelling language (JML). Dafny and Resolve are used to specify contract specification. LML is an extension to Java programming language. Dafny has features of procedural and functional programming languages, whereas Resolver with sound mathematical foundation is used to evaluate contracts and explain later for understanding.

Runtime verification of a webservice is always important to ensure resilience in service rendering. Because, in spite of well-verified design and implementation runtime glitches may creep into put service execution in jeopardy. The issue has been addressed in [61]. Service behaviour is driven by messages. Hence, formalising message passings is the main idea proposed in this paper. Service-oriented description language (SODL) is proposed to specify protocol for message passings. Estelle-based formal model named FSM4WSR is proposed. This model is designed to capture runtime behaviour of a service. This behaviour is validated by the constraints specified in SODL.

Time is a useful and an important feature in case of service provisioning. In practice, a service is required to be done in a specific time. That could be of business interest or domain characteristics. A service execution is seen as a sequence of state changes. And each state has time constraints to be satisfied. For specifying such temporal behaviour [65] proposed a method on timed WS automata. The authors propose methods to translate description of webservice applications written in BPEL4WS to WS Timed Automata. For formal verification, tool Uppaal is used which evaluates time-related properties of a system at different states. The paper explains the proposed method by a case study on an airline travel reservation system.

Dranidis et al. [66] proposes a scheme for runtime verification of webservices. It checks whether a service is running according to specification or not. Primarily, two aspects they verify, that is, control flow and values to variables. During execution, variables must assume expected values. So, control flow must also progress as per specification, i.e. on an emanating context, control must progress resulting in successful termination of service execution. Specification on expected service behaviour is formalised by X-machines. A conversational behaviour is formalised by X-machine which is a kind of extended finite state machine. The machine also helps in monitoring verification process through model animation. The contribution of this paper includes both a method for verification and an architecture for the purpose.

Another work on runtime verification is reported in [67]. The importance of it is verification of temporal properties of a service choreography. Real-time requirements of service interactions are specified in Fiacre verification language. This language is useful to model both behavioural properties as well as timed interactions. Though in a sense the mechanism of verification is the same like other such works, still the speciality is in specifying complex requirements with the help of different formalisms. Logic-based formalism is used to express relation between two occurrences of events. The logic is built on metric interval temporal logic (MITL), a real-time extension of linear temporal logic. This enables verification of very general execution scenario, stated with both local as well as global constraints based on contexts evolving during execution.

A work that is the first in kind reported in [68] takes up both functional as well as nonfunctional requirements verification. In order to verify functional requirements a labelled transition system (LTS) from BPEL semantics is used. The system is drawn from BPEL semantics. For nonfunctional requirement verification, a roundabout strategy is used. Nonfunctional properties are translated to functional verification framework. Three nonfunctional properties, viz. availability, time and cost are considered in the work. The verification of time and cost is performed on the fly. At each state of LTS, availability and cost are calculated and tagged for verification, whereas for response time, at the LTS generation stage required response time is tagged to each state for online verification. Thus, both functional and nonfunctional requirements are verified on the fly. Thus, this is the first work at that time to take up verification of both kinds of requirements. The speciality is that the verification does not require to build a runtime intermediate abstraction to match with LTS extracted from BPEL semantics. Rather, at runtime, the extracted LTS is used for verification.

Yin et al. [69] propose a verification technique that is based on Martin-Löf type theory (MTT). Unlike other formalisation, MTT is not based on predicate logic. Instead, MTT represents predicates as correspondence between propositions and sets. A proposition is identified with a set of its proofs. A set of rules determines a proposition. Existence of a proof ensures proposition. The idea is extended for service matching. Service matching is a proposition that comes true for two compatible services. This compatibility check is governed by type checking rules. In case rules for two services satisfy then it positively verifies composition of two services. The paper discusses on verification of a service and a set of services. The verification is governed by rules for subtype, duality and consistency check.

Table 3 A summary of service verification approaches

Approach	Objective	References
FSM	Bounded message queue in communication service module verification	[52]
CPNET	Protocol conformance—liveliness, boundedness and reachability analysis	[53]
CPNET	Concurrency, communication and synchronisation verification	[54]
VERBUS	Verification of invariants, goals, pre- and post-conditions, activity reachability analysis and temporal properties	[55]
Logic based	Formulating right constraints	[54]
UML (component sequence and state diagrams)	Dependency and correctness tests	[56]
UML (MSC & FSP)	Process and partners roles analysis, message passing, sequence, loops and concurrency verifications	[57]
Petrinet	Control flow verification Static analysis	[58]
Times input and output transition system (TIOTS)	Deadlock detection and protocol matching	[59]
LOTOS and TTCN (behaviour tree)	Safety, liveness and fairness	[60]
CPNET	Reachability, termination criteria, message passing, buffering, choice making tests	[61]
Symbolic model checking	Control and operational behaviour testing	[62]
Event calculus	Pre- and post-conditions checking	[63]
Contract based (JML)	Data consistency check	[64]
X-machines	Control flow, values to variables checks	[66]
Logic based (MITL)	Temporal properties, general execution scenario analysis	[67]
LTS	Functional and nonfunctional requirements verification	[68]

The research works discussed in this section are summarised in Table 3 given below. It presents the verification objectives and associated approaches used for the purpose.

4 Conclusion

This chapter presents some early works on modelling and verification of webservices. The works are representatives of different approaches used for both modelling and verification. In this chapter, FSM, Petrinet, logic and UML are chosen for discussion based on their popularity among users and rigour for the purpose of modelling. Each one is found suitable for modelling of certain aspects of webservices.

FSM, Petrinet, logic, UML, behaviour trees, etc. are used to verify several dynamic properties of service provisioning. The service properties under study include control flow analysis and checking of system liveliness, boundedness, reachability, concurrency, constraints, deadlock, safety, fairness, data consistency, termination and temporal properties. The detailed review gives readers an insight into considering a formal model as a choice for webservice modelling and verification. Readers can appreciate the beauty of each model in specifying certain aspects of an application. So that, for a complex service requirement a designer can choose a set of relevant models useful to model various aspects of an application. At the initial stage of service engineering, such a model composition approach helps to explore uncharted dimensions of an application so, requirement engineering leads towards completeness.

All these modelling approaches being rigorous help in developing concrete algorithms for checking correctness and completeness of service models. It also helps in designing a provable service system as its behaviour is formally defined and realised in implementation.

Further, because of formal approaches for both modelling and verification of webservices, automation of both the processes has been possible. There have been several tools for formal modelling and verification. These tools with respect to different modelling approaches are also discussed.

References

1. Irum Rauf. Muhammad Zohaib Z Iqbal, Zafar. I. Malik UML based Modeling of Web Service Composition- A Survey; Sixth International Conference on Software Engineering Research, Management and Applications, 2008
2. Christophe Dumez, Ahmed Nait-Sidi-Moh, Jaafar Gaber, Maxime Wack, Modelling and specification of Web services composition using UML-S, 4th international conference on Next Generation Web Services Practices (NWeSP08), Oct 2008, Seoul, South Korea. IEEE Computer Society, 0, pp. 15–20

3. Dhikra Kchaou, WS-UML: A UML Profile for Web Service Applications, ISIICT'09 Proceedings of the Third international conference on Innovation and Information and Communication Technology, British Computer Society Swindon, UK, 2009
4. Pengcheng Zhang, Henry Muccini; Model and Verification of WS-CDL based on UML Diagrams; International Journal of Software Engineering and Knowledge Engineering Vol. 20, No. 8 (2010) 1119–1149
5. M. Emilia Cambronero J. Jose Pardo Gregorio Diaz Valentin Valero; Using RT-UML for Modelling Web Services; SAC07 March 11–15, 2007, Seoul, Korea
6. Viet-Cuong Nguyen, Xhevi Qafmolla, Karel Richta; Domain Specific Language Approach on Model-driven Development of Web Services; Acta Polytechnica Hungarica Vol. 11, No. 8, 2014, 121–138
7. W. Provost, "UML for Web services", XML.com, August 5, 2003
8. Jun-Jang Jeng, Wang-Chuan Tsai; Designing An FSM Architectural Framework for Service-Based Applications; COMPSAC, 2000, pp. 234–239
9. R. Gronmo, I. Solheim, "Towards Modeling Web Service Composition in UML", INSTICC Press, 2nd International Workshop on web services: Modeling, Architecture and Infrastructure, Porto, Portugal, April 2004
10. F. Belouadha and O. Roudiés, « Vers un modèle de spécification pour la composition de services web », Proceedings of SIIE'08, Tunisie, Février 2008
11. G. Ortiz and J. Hernandez, "Toward UML Profiles for web services and their Extra-functional Properties", IEEE International Conference on Web services (ICWS'06), 2006
12. F. Belouadha and O. Roudiés, "Un profil UML de spécification de services web composites sémantiques", CARI 2008-MAROC, pp. 537–544
13. Jia Zhang, Carl K. Chang, Jen-Yao Chung, Seong W. Kim; WS-Net: A Petri-net Based Specification Model for Web Services; Proceedings of the IEEE International Conference on Web Services (ICWS04)
14. Yudith Cardinale, Joyce El Haddad, Maude Manouvrier, Marta Rukoz; Web Service Composition Based on Petri Nets: Review and Contribution; LNCS 8194, pp. 83–122, 2013
15. Sofiane Chemaa, Faycal Bachtarzi, Allaoua Chaoui; A high-level Petri net based approach for modeling and composition of web services; Procedia Computer Science 9 (2012) 469–478
16. Y. Deng, S. K. Chang, J. C. A. De Figueiredo, A. Psrkusich, Integrating software engineering methods and petri nets for the specification and prototyping of complex information systems, in: Proc. The 14th International Conference on Application and Theory of Petri Nets, Chicago, 1993, pp. 206–223
17. Hamadi, R., Benatallah, B.: A Petri net-based Model for Web Service Composition. In: Proc. of the 14th Australasian Database Conf., ADC 2003, vol. 17, pp. 191–200
18. Zhang, Z., Hong, F., Xiao, H.: A colored petri net-based model for web service composition. Journal of Shanghai University (English Edition) 12, 323–329 (2008)
19. Yu, H., Fan, G., Chen, L., Liu, D.: Analyzing time constrained service composition based on Petri net. In: 3rd Int. Symposium on Electronic Commerce and Security Workshops, pp. 68–71 (2010)
20. Fang, X., Jiang, C., Fan, X.: Independent global constraints for web service composition based on GA and APN. In: Proc. of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation, GEC 2009, pp. 119–126 (2009)
21. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
22. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WSBPEL processes using flexible model generation. Data Knowl. Eng. 64(1), 38–54 (2008)
23. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Formal semantics and analysis of control flow in WS-BPEL. Sci. Comput. Program. 67(2–3), 162–198 (2007)
24. Martens, A.: Analyzing Web Service Based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)

25. Xiong, P., Fan, Y., Zhou, M.: A Petri Net Approach to Analysis and Composition of Web Services. IEEE Transact. on Systems, Man, and Cybernetics, Part A 40(2), 376–387 (2010)
26. Du, Y., Li, X., Xiong, P.: A Petri Net Approach to Mediation-aided Composition of Web Services. IEEE Transactions on Automation Science and Engineering (2012) (to appear)
27. Li, X., Fan, Y., Sheng, Q.Z., Maamar, Z., Zhu, H.: A Petri Net Approach to Analyzing Behavioral Compatibility and Similarity of Web Services. IEEE Trans. on Systems, Man, and Cybernetics, Part A, 510–521 (2011)
28. Tan, W., Fan, Y., Zhou, M.: A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. IEEE T. Automation Science and Engineering 6(1), 94–106 (2009)
29. Chi, Y.-L., Lee, H.-M.: A formal modeling platform for composing web services. Expert Syst. Appl. 34(2), 1500–1507 (2008)
30. Ding, Z., Wang, J., Jiang, C.: An Approach for Synthesis Petri Nets for Modeling and Verifying Composite Web Service. J. Inf. Sci. Eng. 24(5), 1309–1328 (2008)
31. Yang, Y., Tan, Q., Xiao, Y.: Verifying web services composition based on hierarchical colored petri nets. In: Proc. of the 1st Int. Workshop on Interoperability of Heterogeneous Information Systems, IHIS 2005, pp. 47–54 (2005)
32. Yang, Y., Tan, Q., Xiao, Y., Liu, F., Yu, J.: Transform BPEL workflow into hierarchical CP-nets to make tool support for verification. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 275–284. Springer, Heidelberg (2006)
33. Dong, Y., Xia, Y., Sun, T., Zhu, Q.: Modeling and performance evaluation of service choreography based on stochastic petri net. JCP 5(4), 516–523 (2010)
34. Mao, C.: Control Flow Complexity Metrics for Petri Net-based Web Service Composition. Journal of Software 5(11), 1292–1299 (2010)
35. Xia, Y., Liu, Y., Liu, J., Zhu, Q.: Modeling and performance evaluation of bpel processes: A stochastic-petri-net-based approach. IEEE Trans. on Systems, Man, and Cybernetics, Part A 42(2), 503–510 (2012)
36. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: Compositional Specification of Web Services Via Behavioural Equivalence of Nets: A Case Study. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 52–71. Springer, Heidelberg (2008)
37. Thomas, J.P., Thomas, M., Ghinea, G.: Modeling of web services flow. In: IEEE Int. Conf. on E-Commerce (CEC), Stillwater, OK, USA, pp. 391–398 (2003)
38. Valero, V., Macià, H., Pardo, J.J., Cambronero, M.E., Díaz, G.: Transforming Web Services Choreographies with priorities and time constraints into prioritized-time colored Petri nets. Sci. Comput. Program. 77(3), 290–313 (2012)
39. Blanco, E., Cardinale, Y., Vidal, M.-E.: Aggregating Functional and Non-Functional Properties to Identify Service Compositions, pp. 1–36. IGI BOOK (2011)
40. Li, B., Xu, Y., Wu, J., Zhu, J.: A petri-net and qos based model for automatic web service composition. Journal of Software 7(1), 149–155 (2012)
41. Qian, Z., Lu, S., Xie, L.: Colored Petri Net Based Automatic Service Composition. In: Proc. of the 2nd IEEE Asia-Pacific Service Computing Conf., pp. 431–438 (2007)
42. Cardinale, Y., El Haddad, J., Manouvrier, M., Rukoz, M.: CPN-TWS: a coloured petri-net approach for transactional-QoS driven Web Service composition. IJWGS 7(1), 91–115 (2011)
43. Cardinale, Y., Rukoz, M.: Fault Tolerant Execution of Transactional Composite Web Services: An Approach. In: Proceedings UBICOMM, Lisbon, Portugal, pp. 1–6 (2011)
44. Cardinale, Y., Rukoz, M.: A framework for reliable execution of transactional composite web services. In: MEDES, pp. 129–136 (2011)
45. Mei, X., Jiang, A., Li, S., Huang, C., Zheng, X., Fan, Y.: A Compensation Paired Net-based Refinement Method for Web Services Composition. Advances in Information Sciences and Service Sciences 3(4), 169–181 (2011)
46. Mei, X., Jiang, A., Zheng, F., Li, S.: Reliable Transactional Web Service Composition Using Refinement Method. In: Proc. of the 2009 WASE Int. Conf. on Information Engineering, ICIE 2009, vol. 01, pp. 422–426 (2009)

47. Wang, Y., Fan, Y., Jiang, A.: A paired-net based compensation mechanism for verifying Web composition transactions. In: 4th International Conference on New Trends in Information Science and Service Science (NISS), pp. 1–6 (2010)
48. Sukhamay Kundu; Modeling Complex Systems by A Set of Interacting Finite-State Models; APSEC, 2003, pp. 380–389
49. Basit Shafiq, Soon Chun, Jaideep Vaidya, Nazia Badar, Nabil Adam, Secure Composition of Cascaded Web Services, 8th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing, Collaboratecom 2012 Pittsburgh, PA, United States, October 14–17, 2012, 137–147
50. Andreas Wombacher, Peter Fankhauser, Bendick Mahleko; Matchmaking for Business Processes based on Choreographies; Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service
51. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition; WWW2003, May 2024, 2003, 403–410
52. Xiang Fu, Tevk Bultan, Jianwen Su; Analysis of Interacting BPEL Web Services; WWW2004, May 1722, 2004, New York, USA
53. Xiaochuan Yi and Krys J. Kochut, A CP-nets-based Design and Verification Framework for Web Services Composition, Proceedings of the IEEE International Conference on Web Services (ICWS04)
54. Yabei Wang, Shangliang Pan; CPN-Based Verification of Web Service Composition Model, Proc. International Conference on Educational and Information Technology, 2010, V1-153-158
55. Jesus Arias Fisteus, Luis Sanchez Fernandez, Carlos Delgado Kloos, Applying model checking to BPEL4WS business collaborations, Proc. of ACM Symposium on Applied Computing, 2005, 826–830
56. Zhang, Bixin Li, Henry Muccini, Yu Zhou, Mingjie Sun, Data-enriched Modeling and Verification of WS-CDL Based on UML Models; Zhang, Bixin Li, Henry Muccini, Yu Zhou, Mingjie Sun; Proc. on IEEE International Conference on Web Services, 2008, 752–754
57. Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer, Model-based Verification of Web Service Compositions
58. Chun Ouyang, Eric Verbeek, Wil M.P. van der Aalst, Stephen Breutel, Marlon Dumas, and Arthur H.M. ter Hofstede, Formal Semantics and Analysis of Control Flow in WS-BPEL; Technical Report, Faculty of Information Technology, Queensland University of Technology, Australia
59. Melliti Tarek, Celine Boutrous-Saab, Sylvain Rampacek; Verifying correctness of Web services choreography, Technical Report, IBISC, University of Evry, France … Find reference details
60. Huiqun Zhao, Jing Sun, Xiaodong Liu; A Model Checking Based Approach to Automatic Test Suite Generation for Testing Web Services and BPEL, Proc. of IEEE Asia-Pacific Services Computing Conference, 2012, 61–69
61. Zhuqing Li, Dianfu Ma, Yongwang Zhao, Jing Li, Qing Yang; FSM4WSR: A Formal Model for Verifiable Web Service Runtime; Proc. of IEEE Asia-Pacific Services Computing Conference, 2011, pp. 86–93
62. Quan Z. Sheng, Zakaria Maamar, Lina Yao, Claudia Szabo, Scott Bourne, Behavior modeling and automated verification of Web services, Inform. Sci. (2012)
63. Walid Gaaloul, Sami Bhiri, and Mohsen Rouached, Event-Based Design and Run-time Verification of Composite Service Transactional Behavior, IEEE Transactions on Services Computing, vol. 3, no. 1, January-March 2010, 32–45
64. Iman Saleh, Gergory Kulczycki, M. Brian Blake, Yi Wei; Formal Methods for Data-Centric Web Services: From Model to Implementation; Proc. of IEEE 20th International Conference on Web Services, 2013, pp. 332–339
65. Jia Mei, Huaikou Miao, Qingguo Xu, Pan Liu; Modelling and Verifying Web Service Applications with Time Constraints; Proc. 9th IEEE/ACIS International Conference on Computer and Information Science, 2010, 791–795
66. Dimitris Dranidis, Ervin Ramollari, Dimitrios Kourtesis; Run-time Verification of Behavioural Conformance for Conversational Web Services; IEEE European Conference on Web Services Proc. of Seventh IEEE European Conference on Webservices, 2009, 139–147

67. Nawal Guermouche, Silvano Dal Zilio; Towards Timed Requirement Verification for Service Choreographies; 8th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing, Collaboratecom 2012 Pittsburgh, PA, United States, October 14–17, 2012, 117–126
68. Manman Chen, Tian Huat Tan, Jun Sun, Yang Liu, Jun Pang, and Xiaohong Li; Verification of Functional and Non-functional Requirements of Web Service Composition; LNCS 8144, 2013, 313–328
69. YuYu Yin, JianWei Yin, Ying Li, ShuiGuang Deng; Verifying Consistency of Web Services Using Type Theory; Proc. of IEEE Asia-Pacific Services Computing Conference, 2008, 1560–1567

Webservice Specification and Discovery

Supriya Vaddi and Hrushikesha Mohanty

Abstract Recent advances in Internet technologies have populated the web with a large number of services. Service specification is the first step in SOA for implementation of service publication, service discovery, service selection, and composition. Several specification standards were developed considering the different features of services, viz., service operations, input/output, Quality of Service (QoS), etc. Services are specified in chosen specification standard and are published over repositories/web. The published services are discovered by users on querying the repositories with the specific APIs that are made available or by browsing the web with queries. This chapter gives an overview of the existing specification standards and their features and further, it discusses various approaches of service discovery.

1 Introduction

Webservices have evolved over the years. Earlier, the businesses were confined to a group of organizations. Invention of Internet has made conducting business over web even more simple. Internet is not only used for advertising about a company and its products but to actually perform business over the web. A business could take an order online and deliver the actual product to the consumer at their doorstep. Though this looks simple a lot of concerns are to be addressed from both consumer point of view and from service provider point of view. As both service provider and consumer are not associated with each other, the major concerns that are to be addressed would be (i) How does a consumer know about a business? (ii) How can a business tell

S. Vaddi (✉)
School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India
e-mail: supriyavaddi@gmail.com

H. Mohanty
Kalinga Institute of Industrial Technology, Deemed to be University, Bhubaneswar 751024, India

H. Mohanty
School of Computer & Information Sciences (On leave), University of Hyderabad, Hyderabad, India

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_2

about its details without revealing the business logic? (iii) A business in order to process a user order has to communicate with various other applications. How can applications that are technically different communicate with each other?

Service-Oriented Architecture (SOA) has evolved as a unanimous paradigm providing solutions to the above-addressed issues enabling development of distributed applications.

Service is the core element of SOA. Each service is a bundled business available over web that can be availed by a consumer. These bundled businesses are webservices and are the core element of SOA. A webservice is a self-describing self-contained modular application that can be described, published, located and invoked over a network.

Unlike the standard software that is installed on systems of different consumers. A webservice is readily available on the web and a consumer in need of a service has to invoke the service with a set of inputs. The prominent features of webservices are reusability and composability.

Any number of consumers can invoke the same service with different inputs any number of times as they are reusable. Services being loosely coupled can be invoked irrespective of their working platforms. And they cooperate among themselves to deliver a desired service.

A webservice uses distributed environment in which applications and components can interoperate in a manner independent of their implementations (e.g., platform independent, language independent).

The webservice architecture in SOA paradigm is shown in Fig. 1. It provides a means for service providers and service consumers to know about each other. SOA is a registry-based architecture.Universal Description Discovery and Integration (UDDI) [1] is a universal business registry which acts as a central repository in SOA architecture. All webservices offered by service providers are registered with registry UDDI. The registry stores details of publisher, published service, and descriptions of services as shown in Fig. 1. Service descriptions are made in specification standard called Web Services Description Language (WSDL) proposed by W3C. These service interfaces coded in WSDL point to actual services that are to be invoked by a service consumer.

Based on requirement service consumer would search for services in UDDI. Later, service interface details are followed up and a particular service is selected to be consumed. The services provided could be implemented in any language (C, C#, Java…) and is independent of service registry implementation. The selected service is invoked using binding details that are published on registry.

In the case of webservices, unlike traditional software engineering approach, software modules are not built on gathering requirements from the consumer. But rather requirements are mapped to services that are already built. Hence, the process of service discovery from consumer and provider perspectives is different. The generic process of discovery from each of these perspectives is shown in Fig. 2. Here, a service provider would specify a service in specification format that registry is expecting.

The standard service specification formats are WSDL [2], OWL-S [3], SAWSDL [4], and WSMO [5]. The specified services are stored either in database or as a flat

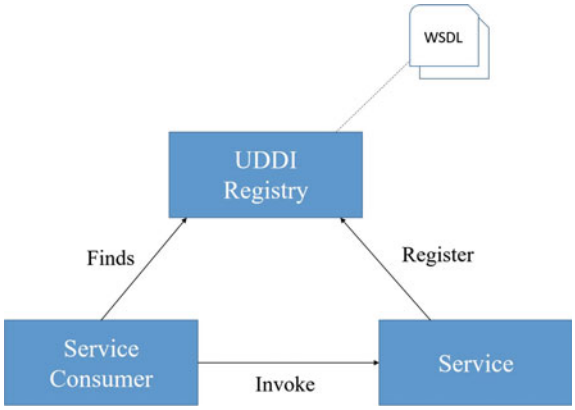


Fig. 1 Webservices architecture

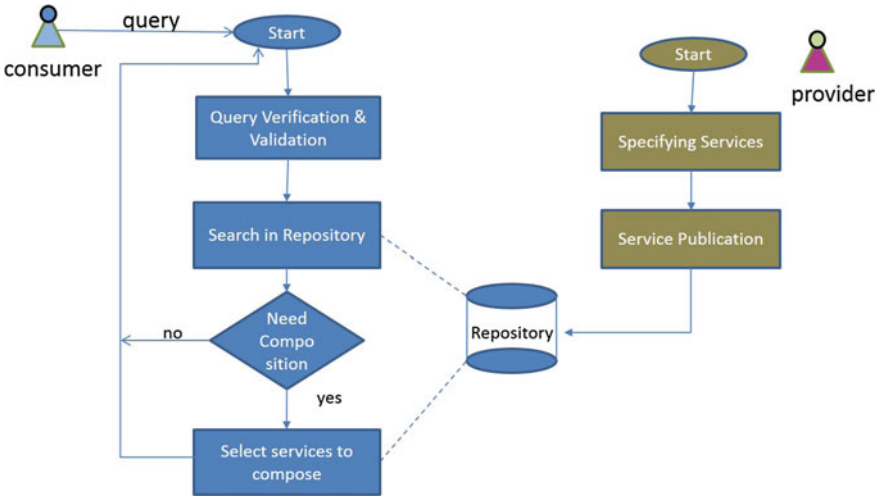


Fig. 2 Service discovery from service consumer and provider perspectives

file in XML. A user in need of a service would query a repository with requirements in the specified format. On verification of query, repository is searched to retrieve all services that meet the query. If none of the available services from the repository is meeting user requirement, then a set of services are to be composed to meet the requirements.

A service provider would tell about its service following a definite specification so that service details are expressed in a standard form that makes users convenient to browse, and also helps other systems to interface without any integration incompatibility problem. A detailed study of service specification standards is discussed in Sect. 2. Searching of a service from repositories is discussed in Sect. 3.

2 Webservice Specification Standards

Specification (Dictionary meaning) is a detailed description of how something is, or should be, designed, or made. Basically, a service specification gives a detailed description of how a service is designed. According to IBM, a service specification must specify everything that a potential consumer of the service needs to know to decide if they are interested in using the service, as well as exactly how to use it.

The basic information a specification on a service has to provide are (i) Name of the service, indicating what the service is about. (ii) Provided and required interfaces, describing the functional capabilities of the service. Functional capabilities include function name, required, or optional service data inputs and outputs, preconditions that consumers are expected to meet before using the capability, exceptions, or fault conditions that might be raised if the capability cannot be provided for some reason. (iii) Communication protocol or rules that determine when the capabilities can be used or in what order. (iv) Qualities that service consumers should expect and that providers are expected to provide such as cost, availability, performance, footprint, suitability to the task, competitive information, etc. (v) Policies for using the service such as security and transaction scopes for maintaining integrity or recovering from the inability to successfully perform the service or any required service.

Initially, webservice specification was standardized by IBM and Microsoft and Web Services Description Language (WSDL 1.1) was published in March 2001. Later, World Wide Web Consortium (W3C) has worked on specification and released WSDL 1.2 in the year 2003. According to W3C, "WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint". WSDL [2] is used to describe service interface. It specifies services as a collection of network endpoints or ports associated to operation names, data format of the Input (I) and Output (O) are described as message types.

WSDL 2.0 component model is given in Fig. 3. A service is availed by either sending or receiving a sequence of messages. The message format is declared under type's component of the model. Each message may be simple with one element or can be complex with many elements. Each of these elements and their data types is declared under an element.

For example, address element is complex and contains house number, streetname, city, state, and pin code. Each of these elements data types is declared house-no: String, Streetname: String, city: String, and state: String and pin: Int. If a schema file containing all these details is present, then that particular xsd (XML schema definition) is imported in types. The Following is the syntax of example for declaring the complex type message address.

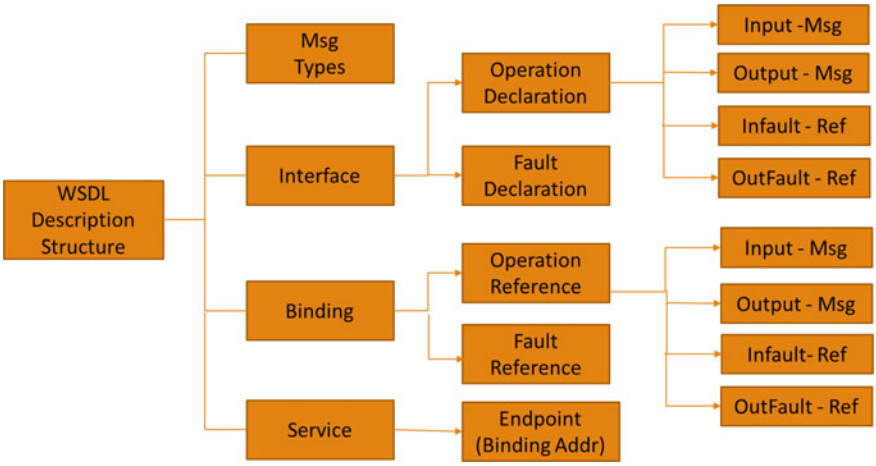


Fig. 3 Component model of WSDL

```
<xs:element name="address" type="tAddress"/>
<xs:complexType name="tAddress">
<xs:sequence>
        <xs:element name="houseNumber" type="xs:string"/>
        <xs:element name="streetName" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="pin" type="xs:integer"/>
</xs:sequence>
</xs:complexType>
```

Communication with service takes place through a set of operations defined in the interface. An interface has a name and it can be an extension of other interfaces by using extends attribute. All communications supported by an interface are defined in operations. Each of these operations has input and output messages. A message communication may result in an error when the message fails either to get transmitted or received. WSDL supports handling of these faults by declaring fault component for each operation. A fault message may be used to communicate information, viz., the reason for the error, the origin of the fault, as well as other informal diagnostics such as a program stack trace. Hence, operations not only define ordinary message interactions but also fault messages that are exchanged between service and users of service. Also, operations define message exchange pattern, viz., number of messages and the order of exchange of these messages. There are eight message exchange patterns (MEP) defined in WSDL 2.0 considering the faults. These patterns are discussed below, indicating the direction of message flow with in or out, respectively.

Fig. 4 Syntax for operation specification

```
<description>
<interface>
<operation
        name="xs:NCName"
        pattern="xs:anyURI"?
        style="list of xs:anyURI"?
          wsdlx:safe = "true">
<documentation />*
    [ <input /> | <output /> | <infault /> |
<outfault /> ]*
</operation>
</interface>
```

(1) **In-Only**: This pattern has only one message that is received (in) by the service.
(2) **Robust In-Only**: This pattern has exactly one message that is received (in) by a service. On the occurrence of fault, it returns a fault message.
(3) **In-Out**: This pattern has two messages and follows the sequence. The service receives a message (in) and returns a response message (out).
(4) **In-Optional-Out**: This pattern has one or two messages in sequence. The service receives a message and optionally returns a response message.
(5) **Out-Only**: This operation has only one message and service sends (out) a message.
(6) **Robust-Out-Only**: This operation has only one message. The service sends (out) a message and on the occurrence of fault at the consumer side (or third-party service), it receives a fault message.
(7) **Out-In**: This pattern has two messages in sequence. The service sends (out) a message to the consumer and in turn, receives (in) a response message from the consumer.
(8) **Out-Optional-In**: This pattern has one or two messages in sequence. The service sends (out) a message and receiving a response message is optional.

For patterns In-Out and Out-In consisting of follow-up message if the fault occurs after single message communication, then instead of the next message in sequence fault message would follow. The syntax for operation is shown below in Fig. 4.

Each operation is specified by a unique name and the pattern is an URL referring to one of the eight MEPs defined in WSDL 2.0.

An optional attribute style indicates the type of communication protocol that is chosen for implementing the pattern. Style is of three types such as RPC, IRI and the Multipart each applying different set of rules. If the operation meets the criteria of safe interaction, then wsdlx:safe attribute is marked as true else false. This is followed by fault or input/output messages.

```
<description>
<interface name=”ShoppingCart”>
<fault name=>
<operation name=”Delivery”
   pattern=”http://www.w3.org/ns/wsdl/out-in”>
<output
        messageLabel="out"
        element="tns:DeliverItems">
</output>
<input
        messageLabel="In"
        element="tns:DeliverItemsResponse">
</input>
<infault                    ref="tns:InvalidAddress"
messageLabel="In" />
</operation>
</interface>
</description>
```

Binding component gives actual implementation details like message formats and protocol interactions associated with operations and also faults. Binding is given a name which would be referred by service element. Reference to the existing interface element already defined in WSDL is made in interface attribute. Message format of the interface is given in type and binding is made available in multiple transport values http-get, http-post, or SOAP.

```
<description>
<binding
      name="xs:NCName"
      interface="xs:QName"?
      type="xs:anyURI" >
<documentation />*
   [ <fault /> | <operation /> ]*
</binding>
</description>
</interface>
</description>
```

The binding extension is available in SOAP and HTTP to support different versions of each protocol. The declared operations and faults are referred here in binding.

The implemented service is deployed and this information is available as a collection of endpoints in the service component. Each service element is a given unique name and it is an instance of the interface defined previously in interface element.

```
<description>
<service name="xs:NCName" interface="xs:QName" >
<endpoint
        name="xs:NCName"
        binding="xs:QName"
        address="xs:anyURI"? >
<documentation />*
</endpoint>+
</service>
</description>
```

Each endpoint is identified by a name and refers to binding element previously defined and URI points to the actual address where the service is available.

Documentation element throughout the WSDL document contains human-readable or machine-readable content that a service provider or developer gives describing the service. These descriptions would also aid query processor when searching for a service meeting query.

Generally, a service provider and consumer frame service descriptions without knowledge of each other. Ontology emerged as a solution for this. Ontology is a formal explicit description of concepts in a domain of discourse (classes), properties of each concept describing various features and attributes of the concept (roles or properties), and restrictions on roles (role restrictions) [6]. Ontology is introduced for webservices so as to enable different service providers to declare and describe the services using standard terminology with predefined set of classes and properties. Several specifications like SAWSDL, OWL-S, and WSMO are proposed that refer to these ontologies. WSDL 1.2 specification was updated to WSDL 2.0 enabling semantics using these ontologies. Semantics is giving meaning to the words. During service specification concepts that are available as ontologies at standard URIs are used in defining names of operations, messages, etc. These ontologies are specified in Web Ontology Language (OWL) that can be referred by both service provider and consumer.

SAWSDL an extension to WSDL with semantic annotations is proposed in [4] to give the meaning to keywords. SAWSDL does not specify a new language to represent semantic models but refers to external ontologies in the WSDL itself. The data mapping of XML schema types from annotations to ontology is given. FUSION [7] is one of the semantic registries that this work has developed. It uses SAWSDL and OWL knowledge base for service specification.

OWL-S [3] segregates service information into three categories, namely service profile, process model, and grounding as shown in Fig. 5. Each of these categories is described in separate ontologies. Service profile states what a service does and describes functional parameters, i.e., hasInput, hasOutput, and precondition and effect (IOPEs). It also describes nonfunctional parameters such as serviceName, serviceCategory, qualityRating, textDescription, and service provider details like name and contact information.

A process model describes how a webservice performs its tasks. Provision to specify the order of messages exchanged is given thus communication protocol can

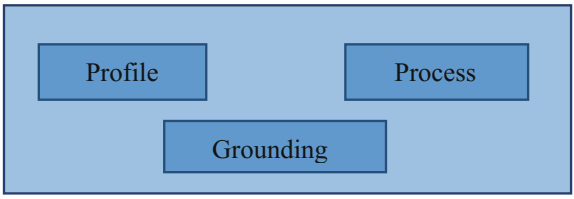


Fig. 5 Components of OWL-S specification

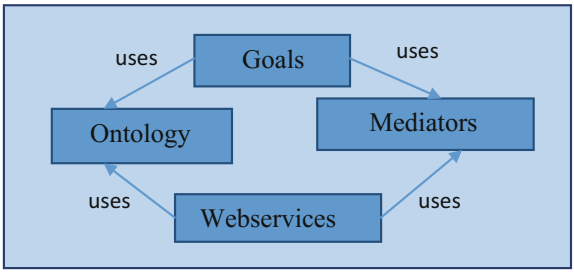


Fig. 6 Components of WSMO specification

be derived from it. Each process model is annotated as atomic or complex. A process model is said to be atomic or simple if message exchange is of request-response type. Whereas, it is said to be complex if more number of message transfers are involved.

Grounding specifies mechanisms to access a webservice, viz., transport protocols, message formats, and other service-specific details such as port numbers used in contacting a service. Further description of each of OWL-S elements is given in the syntax shown in Fig. 7. Though OWL-S has been successful for semantic specification and discovery of webservices based on domain ontology, but there are no standard ontologies. Making an ontology for a domain brings in constraints in specifications, as all are limited to use the ontology that is agreed upon. The attempt has been made to overcome the limitation allowing users to generate new ontology combining the concepts available in ontology (Fig. 6).

WSMO [5] is one such attempt. Web Services Modeling Ontology (WSMO) describes services in terms of four main elements, viz., Ontology's, Webservice descriptions, Goals, and Mediators. Ontology is introduced to maintain same terminology with other WSMO elements descriptions, viz., webservices, goals, and mediators. Webservice element provides a conceptual model for describing webservices nonfunctional properties, capabilities, and interfaces. Goals represent the user's requirements following the same syntax as that of webservices stated before; Mediators deal with interoperability problems that arise between different WSMO elements.

On the whole, WSMO ontology can be used to define the concepts, sub-concepts, super concepts, and similar concepts using Concept and Relation elements. It can be perceived that WSMO is higher in abstraction than OWL-S as it provides a means

Spec Name	Syntax
WSDL	:: <operation, input, output,infault, outfault, BindingDetail, EndpointDetails>
OWLS	:: <profile><process><grounding> <*profile*>:< SrvName, SrvClassification,TextDesc ..> <*process*>:<hasInput,hasOutput,hasPrecondition,hasResult,ComposedOf> ComposedOf:<controlconstruct,sequence, split, split-join, any-order, choice..> ::*Grounding*<WsdlAtomicProcessGrounding > <WsdlAtomicProcessGrounding>:<wsdlDocument,wsdlOperation,wsdlService,wsdlInputMessages,wsdlInput, wsdlOutputMessages,wsdlOutput.>
SAWSDL	:: <ModelReference><LoweringSchemaMapping><LiftingSchemaMapping >
WSMO	:: <Ontology><WebService><Goal><Mediator> <*Ontology*>:<usesMediator, hasConcept,hasRelation,hasFunction,hasInstance,hasAxiom> <*WebService*>:<importsOntology, usesMediator,hasCapbility, hasInterface> <*Goal*>:<importsOntology, usesMediator, requestsCapbility,requestsInterface> <*Mediator*>:<importsOntology,hasSource,hasTarget,hasMediationService>

Fig. 7 Syntax of different service specifications

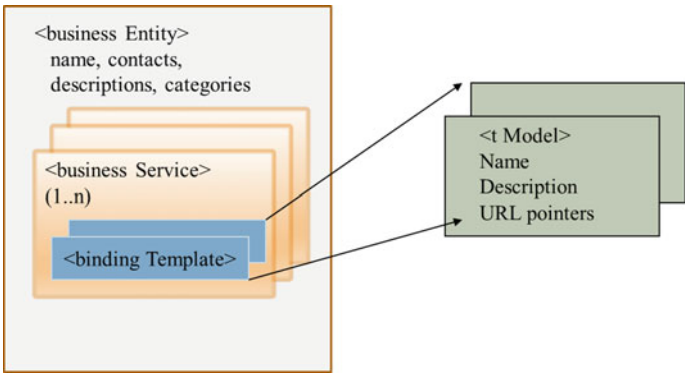


Fig. 8 Core data structure of UDDI

to generate new concept from given ones; thus allowing users unlimited capability for generating newer concepts. This capability while is a boon, also can be a bane for difficulty in tracing concepts and getting a meaning as they are discrete without having defined inter-concept relations. Defining so many concepts and relations makes WSMO heavier. WSMO consortium is working to standardize concept-based service specification. And academia, as well as professionals, is adding new dimensions to WSMO [5, 8]. WSML [9] is a specific designed language for WSMO and contains logical formulae to describe different WSMO elements. Figure 7 provides a comprehensive syntactic description of service specifications following WSDL, OWL-S, SAWSDL, and WSMO formalisms.

A protocol defines an order of exchange of messages between user and provider in availing a service. WSDL gives operation details which is sufficient for using a service that is stateless in nature. A service is said to be stateless when it accepts an input that contains all necessary information to process and generates an output. Whereas, a service is said to be stateful when a service takes some input and messages are exchanged in between before generating output.

Table 1 Comparison of different service specifications

References	Name	In	Out	Cond	Protocol	Ontology
WSDL [2]	☑	☑	☑	–	–	–
SAWSDL[4]	☑	☑	☑	–	–	☑
OWL-S [3]	☑	☑	☑	☑	☑	☑
WSMO [5]	☑	☑	☑	☑	☑	☑

OWL-S [3] and WSMO [5] specifications introduced earlier supported the order in which intermediate communication happens. For this, OWL-S process model would be stated as composite process that is composed of messages with control constructsSequence, Split, Split + Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until. Syntax of four types of standard specifications (WSDL, OWL-S, SAWSDL, and WSMO) reviewed is shown in Fig. 7. A comparison of these on different perspectives is presented in Table 1. Existing specifications concentrate only on input, output and function name and none of this address the issue of service structure and its role in-service specification.

Research work in [10] considers service structure and has proposed Service Map (SMap) for specifying service. A service is viewed as a collection of service items packaged to cater to different needs of users. These service items are connected with associations includes, akindof, leads-to, and has-with and combinations of service items is specified with AND and OR. The service items description is extendible and is not limited by the specification. A service provider can present service features, viz., promotions, conditions, and alternatives that attract the consumer.

Targeting toward service flow and composition service specifications, viz., Web Services Business Process Execution Language (WS-BPEL) [11] and Web Services Choreography Description Language (WS-CDL) [12] have evolved over time. Earlier, Business Process Modeling Language (BPML) [13] was used for specification now it is part of OMG and is replaced by WS-BPEL.

WSDL is the specification standard followed by industry for specifying services. Since the standard is rigid not allowing user to specify more detail WSMO evolved. Though WSMO is very descriptive, it is complex for a service provider to publish a service in WSMO. But the information needed to attract users is not supported. On the other hand, WSMO details technical details that one-way risks service providers of malicious usages and in other way, it also makes consumer burdened for availing a service. As a customer has to go through enormous amount of specification detail to check if a service meets requirement.

Since many services are published, an issue of identifying a service meeting the requirement is of prime importance. Services with different specification standards are available; to search for a service a user should have basic knowledge of specification syntax. Instead of user searching through all the available services, the system provides a means for searching through large number of services and list down services that are meeting user requirements this would reduce burden on the user. Hence

on discussing standards for service specifications, it is natural is discuss on searching of webservices in a service repository. In the next section, we will address the issue.

3 Service Discovery: A Bird's View

The services available on the web are growing exponentially, selecting the most relevant webservice fulfilling user requirements is challenging. Service search (or service discovery) is a process of searching for webservices matching a given set of functional and nonfunctional requirements of users. The services could be published on the web or on registries, and in different specification standards discussed in Sect. 2. Thus, searching for a service is a difficult task and is performed differently for the web and service registries. Each registry has services specified in a specific standard. Various approaches have been used for service search such as searching in UDDI, web and service portals (Registries). Here, we would discuss each of these searches in detail.

3.1 Web Search

Searching web by using different keywords or with combination of keywords is known as web search. In return, search engines would list enormous number of web pages which a consumer has to browse through to identify the needed service. This is a tedious process and to overcome it, automation of service search has received much attention. A technique proposed in [14, 15] shows the preparation of service repository by web crawlers fetching service data on its WSDL interface accessing URLs of services hosted on the web. In [15], Dong et al. have proposed Woogle that categorizes services into a group based on similarity among services. The similarity is found based on semantic matching of parameters that specify services. It means a cluster of services is found suitable for similar services. Thus, work in [14] shows it is important to describe a service with defined syntax and words of proper semantics.

Later, for flexibility in specification, WSDL specification is extended with XML extension, that means a crawler has to search for WSDL of a service in its XML file available at an URL hosted by a service provider. As shown in [16], XML-based service specification is well appreciated for its portability as well as flexibility in extending a specification.

3.2 Directory Search

In directory-based approach, service providers would publish their services in the specific format specified for a directory. A consumer in need of a service would query

the directory. UDDI [1] is one such well-known service repository where services are published and queried using APIs. UDDI has emerged as a model and in similar lines, juddi an open-source implementation is made available on the web. Targeting towards reuse organizations tend to set up juddi locally to publish software modules developed in-house for interorganizational use.

3.2.1 Service Search in UDDI

Universal Description, Discovery, and Integration (UDDI) [1] is an industry standard for service registries, developed to solve the webservice search problem. UDDI enables service providers to register themselves with their business name. On registration, they can advertise services and products offered and specify how the business transactions can be conducted on the web.

The provided data is captured and stored in the information model. A UDDI information model is composed of four primary data structures as shown in Fig. 8:

1. businessEntity: describes a business or service provider that typically provides webservices, viz., business name, contact detail, and other business information.
2. businessService: represents business descriptions for a webservice. It describes a logical collection of related webservices offered by a businessEntity.
3. bindingTemplate: contains the technical information associated with a single webservice for interacting and binding. Each bindingTemplate describes an instance of a webservice offered at a particular network address, typically given in the form of a URL. The bindingTemplate also describes the type of webservice being offered using references to tModels, application-specific parameters, and settings.
4. tModel: technical specification for a webservice is defined in tModel. It points to the URL where the actual specification is present. The services published in UDDI are categorized for easy search the detail of each service category is given in tModel. It also contains the type of protocol (SOAP or HTTP) used for communicating with service.

Search in UDDI is through set of APIs provided for inquiry. These are find_binding, find_business, find_relatedBusiness, find_service, find_tModel, get_bindingDetail, get_businessDetail, get_operationalInfo, get_serviceDetail, and get_tModelDetail.

Existing open web registries for publishing and querying of webservices are WebserviceX (WebserviceX.NET) [17], XMethods (XMethods.NET) [18]. They support search by browsing or by keywords where all the services having the queried keyword in their service description are listed. Searching registry is either done manually by browsing through services or by performing keyword-based search. For browsing, the consumer should have prior knowledge of a service. Searching by keywords is found to be insufficient [15] as published services have very small information for choice of keywords. Further meaningfulness of keywords with respect to services is also a question of concern.

3.3 Service Search Classification

Either search on the web or in registry depends on the information available on the web or at registries. The operations information is present in WSDL documents. Traditional search method is to follow information retrieval approach where required service operations are searched in WSDL documents. As services are exposed by their interfaces, most of the research works in discovering a service focuses on matching Input, Output, and their data types. Further, the search is narrowed down by introducing precondition and effect. While some approaches are based on interface matching, in contrast, other research works consider structure of the service for matching. Ontologies have paved way to semantic webservices. Further with the introduction of ontologies, logic-based approaches have emerged for retrieval of services. Service discovery approaches can be categorized as follows also shown in Fig. 9.

3.3.1 Information Retrieval Approach

Information retrieval approach is a traditional approach of retrieving required document from the collection of documents on examining words that are common to documents. Text document matching is the method of identifying set of documents that contain a given text. Few such works are used to build a vector space model [19] to create an index [20] on collection and query by example. In the first case, each service is represented as a weighted vector of unique words. Each document is parsed for phrases/sentences and these are divided into words. From these, stop words are eliminated and stemming is performed on words to get root word. In the case of webservices, WSDL document describes a service by its keywords. Term Frequency (tf) and Inverse Document Frequency (idf) of words are used to build up the vector space model and in a way to measure the similarity between documents.

In the second case, an index is created using inverse document frequency. In inverse document frequency, the term that occurs in fewer documents is a better discriminator than a term that occurs in most of the documents. Hao et al. [20] uses both information retrieval and similarity between XMLs to select the documents. Each service in XML has a schema tree. Distance between trees gives similarity between services. A variant of tree edit distance is used to measure the distance.

A user, query is represented as weighted vector and is searched among repository of services represented as weighted vector. Service preference is defined in terms of service relevance and service importance. Cosine similarity measure is used to find the relevance of service with query. If many services use (connect to) a service, then the used service is important to the one that is not used. Connectivity between services is obtained by matching schema tree of services. If schemas match connectivity score is high and so is the importance score.

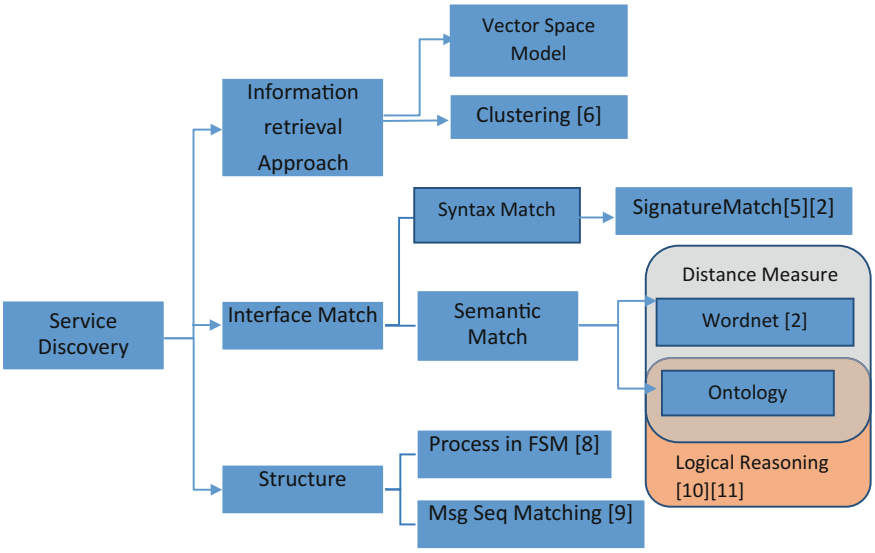


Fig. 9 Classification of service search approaches

In query by example [21], case consumer describes the expected service as an example and this is preprocessed to represent as a vector. The above-stated approaches would be discussed in detail in Sect. 4.

3.3.2 Interface Match

Interface of a webservice has description of service, operations, messages, and their data types as shown in Sect. 2. Matching of interface can be classified into matching of syntax (data type match) and matching of semantics (names) as shown in Fig. 9. Signature matching approach is used for matching syntax. Semantic matching involves matching service names, operations, and message names. Semantics is giving meaning to words. Wordnet and ontologies are two approaches of adding semantics to words. Wordnet [3] is a large lexical database of English connecting meaningfully related words and concepts to form a network. Ontology is introduced for webservices so as to enable different service providers to declare and describe the services using standard terminology with predefined set of classes and properties. Several distance measures are proposed to obtain the distance between words in network and among ontology classes. Most of the research works use combination of semantics and syntax match to compute similarity score between services. Logical reasoning is applied over services using ontologies

Each of these categories is briefly discussed below.

Reusability of software has received a lot of attention in the 90s with Component-Based Software Engineering (CBSE) [22]. Where, Commercially-Off-The-Shelf components (COTS) are built for reuse. Component is a nontrivial, nearly independent and replaceable part of a system that fulfills a clear function in the context of well-defined architecture. In component retrieval from libraries for given desired requirements, a software module meeting the stated requirements is retrieved from library using signature matching [23]. Signature matching is matching the functions type that is a list of input and output parameters. Extensions to this signature matching are applied to webservices [24] for interface matching.

Signature matching for webservices compares the set of operations offered by services. As operations consist of input and output messages, corresponding input messages, and output messages of services are compared. Since each message is described with data types in succession these are compared. Each message can be simple or complex type. A complex message has to be further matched recursively.

For example, reviewing shopping cart service discussed in Sect. 2 from Fig. 4 signature of delivery operation would be as follows:

Operation: Delivery
Input: DeliverItems
Output: DeliverItemsResponse
Messagetype:
Input data type:customerIDint,
orderNumber int,
customerName<*FirstName,MiddleName,LastName*>,
itemList <*ItemID,ItemName,Quantity*>,
address<*houseNumber, streetName, city, pin*>
paymentDetail<pre,post>

Output data type:deliveryDate date,
receiverName string

The operation name is "Delivery". It takes DeliverItems message as input and gives DeliverItemsResponse as output. DeliverItems is a complex message with sequence of message elements customerId, orderNumber, customerName, itemList, and paymentDetail. Each element has simple data type or complex type creating a syntax. Signature matching of this operation corresponds to matching all the message elements according to syntax and their data types.

Syntax matching [25] is an extension of signature matching, though paper presents it as structure matching and we look structure of XML as syntax. Here, WSDL operations, input/output messages, and their data types are compared and matching score is computed. This score also considers matching between parameters that come in different orders.

In semantic matching [25], scores are assigned based on the distance between words in wordnet hierarchy. Querying over collection of WSDLs is also performed with synonyms of terms from requested WSDL, thus expanding the result. Semantic

structure matching score is computed between requested WSDL and WSDLs present in the repository. Top WSDLs with maximum scores are selected from repository and these retrieved services were found to be close to query. A detailed description of interface matching is discussed in Sect. 4.2.

3.3.3 Structure Match

In the above subsection, webservice interfaces are used as references for matching services. But the functionality of service is obtained on the following certain process. In [26], the authors have proposed a matching approach based on the internal process of services. Where, both service internal process and sequence of functionalities expected by requester is modeled as FSM.

For matching, two FSMs are compared and different metrics are used to find similarity score between services. For this, all possible sequences of both queries FSM and service FSM are obtained as strings. Each string of request is compared with all sequences of service. Based on certain distance metrics, viz., Common Process Count (CPC), Longest Common Substring (LCStr), Longest Common Subsequence (LCSeq), and Edit Distance (ED), the structural similarity is calculated. CPC is ratio of number of common processes to the total number of processes in the sequences. LCStr is number of contiguous processes between query and service sequence. It is ratio of length of common substring to length of query string. ED is minimum number of operations of insertion, deletion, or substitution of single character to transform one string to another. Combination of these metrics was found to result in many qualitative results compared to individual metrics alone.

Webservices are described as business processes [27] rather than as operations and associated individual messages. The focus of this paper is in matching the communication sequence of query service and service from repository. The message sequence is obtained from BPEL file given by the provider during the publishing of service. Two services are said to be matched when they have a common message sequence, i.e., their languages have non empty intersection. This match is done by using FSA. This work addresses more of behavior matching than structure matching.

4 Service Search Techniques

For this traditional approach like Vector Space Model as briefed below uses text-matching technique.

4.1 Information Retrieval Approach

4.1.1 Vector Space Model

Vector Space Model (VSM) is a method of representing a document as a vector where unique words present in the document form vector components. So, the vectors from a set of documents make a vector space model. In this approach, the search is performed over collection of text documents. Number of times the terms occur in a document is of importance for referring the document by such words. In the case of webservice discovery, the assumption is that all the available services are specified in WSDL and are stored in a centralized repository preparing a central repository of webservices is the first step in making an architecture that is used for search of services on a request of a service user. While searching for a request, i.e., of keywords in the request, number of services matching the keywords with service vectors are picked up.

Here, we would brief on the concept of representing document as vector. Various weighing factors viz. Term frequency (tf), combination of term frequency and inverse document frequency (tf-idf) could be used for a term in document.

Term frequency (tf_{ij}) of a term i in document j is total number of occurrences of term i in j.

Document frequency (df_i) is defined as the number of documents in collection N that contain the term i. Inverse document frequency (idf_i) of term i in collection of documents N is defined as

$$idf_i = \log \frac{N}{df_i}$$

Term frequency and inverse document frequency determines the weight of vector (W_{ij}) associated with term "i" in document "j".

$$W_{ij} = tf_{ij} \times idf_i = tf_{ij} \times \log \frac{N}{df_i}$$

The document is now represented as vector with terms from dictionary and each term is weighted with weights given by the above equation. If a term is not present in a document, then the corresponding weight for that term is zero.

If $\{t_1, t_2, t_3, t_4, \ldots, t_n\}$ are terms in dictionary. Then, the vector associated with document (d_1) containing terms $\{t_2, t_4, t_7, t_9\}$ is

$$\overrightarrow{V_1} = w_{21} * t_2 + w_{41} * t_4 + w_{71} * t_7 + w_{91} * t_9$$

A query with a set of terms is represented as a vector. Searching for a query document in the collection of documents is done by measuring similarity between the query document and documents in collection. Similarity measure is a function which computes the degree of similarity between a pair of vectors or documents. Several

Table 2 Example of weighing WSDLs and query by term frequency

Doc/term	add	basket	deliver	item	product	remove	save
x.wsdl	1				3	1	1
y.wsdl	1	2				1	
z.wsdl	1		1	4		1	1
Q	1		1	3		1	

Table 3 Similarity of query to WSDL documents

Query/document	x.wsdl	y.wsdl	z.wsdl
Q	2/12 (0.166)	$2/\sqrt{72}$ (0.166)	$3/\sqrt{240}$ (0.968)

similarity measures, viz., Euclidean distance, cosine similarity, Jaccard coefficient, Dice coefficients, and Pearson correlation coefficients [28] are applied to find similarity score between query vector and collection of document vectors.

Vector space search engine [29] an extension to UDDI has been proposed to search and index webservices. Here, WSDL documents are collected from reference specified in UDDI are preprocessed and VSM is built. In preprocessing, keywords are extracted from endpoint URL, attribute names, message names, and XML comments of each WSDL. To search, a given query is represented as a vector and cosine value is evaluated. Cosine value between two vectors p and q is calculated as

$$\cos(p, q) = \frac{p \cdot q}{|p||q|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2 \sum_{i=1}^{n} q_i^2}}$$

where p · q is dot product between the vectors and $|p|\, and\, |q|$ is Euclidean distance. For documents with high similarity, cosine value tends toward 1 and with less similarity value tends toward 0. Sorted list of services (WSDLs) based on their similarity score is shown as result.

For example, consider a user in search of a shopping cart service with operations addItem, removeItem, and deliverItem. Assume that repository contains three shopping cart services x, y, and z. Each having operations as x(addProduct, removeProduct, and viewProduct), y(addToBasket, removeFromBasket), and z(addItem, removeItem, saveItem, and deliverItem). In preprocessing, terms are extracted and frequency of occurrence of each term is noted as shown in Table 2.

Cosine similarity measure is applied to compute distance between query and document vectors. Computed cosine value is shown in Table 3. It can be observed that service z has high cosine value it implies that z is more similar to query compared to other services (x, y).

VSM model has been used by Wang and Stroulia [25] for searching services that are specified in natural language under WSDL description. It uses both information and component retrieval approaches so as to enable programmatic service discovery. For this, preprocessing is performed over the documents to remove stop words and

to combine related words into common word stem. The model is powered with wordNet [30] to have subvectors. Each of them includes (i) stems of original words, (ii) synonyms, and (iii) direct hypernyms, hyponyms, and siblings for the terms in document and for all word senses. For a given query, initially, VSM is used to identify a set of services from a collection and over this pruned list [25], semantic structure matching is performed to match operation and message names.

Structure matching is an extension of signature matching where WSDL operations, input/output messages, and their data types are compared and matching score is computed. This score also considers matching between parameters that come in different orders. In semantic matching, scores are assigned based on the distance between words in wordnet hierarchy. Querying over a collection of WSDLs is also performed with synonyms of terms from requested WSDL, thus expanding the result. Semantic structure matching score is computed between requested WSDL and WSDLs present in the repository. Top WSDLs with maximum scores are selected from repository and these retrieved services were found be close to query.

In another work [31] Preference degree of a service is proposed for ranking services. Service relevance and service importance are two desired properties for preference degree. Service relevance is obtained by measuring the tf/idf of list of words in WSDLs of repository. Service importance identifies the relation between services based on schema matching. Service connectivity is measured by using schema tree-matching algorithm.

Computing distance between query vector and vector associated with the published service will be time-consuming as repository size increases. Instead of performing search over the entire repository if available services are clustered and search is performed on one of the clusters, the search space is reduced. Deciding on what basis clustering is performed is important as similar services are expected to be grouped together.

4.1.2 Clustering

Information retrieval approach has also been used in Web Service Query By Example (WSQBE) [21]. This work follows a two-step approach for retrieving matching services. In the first step, search space is reduced by identifying a group of similar services and in second step for given query, subspace is searched and ranked list of services are obtained. The first step of WSQBE uses automatic document classifier to classify services of a repository into clusters based on their contents. The consumer would give an example describing an interface with set of words or gives functional description of expected operations. The example is preprocessed to remove irrelevant words and resulting set of words are represented as a vector. In the next stage, this example vector is classified into one of the existing categories. Now, cosine similarity score is computed between the query vector and service vectors to retrieve matching services. Top 50% of services are shown as candidate services for the query.

In [15], Dong et al. have proposed Woogle that supports service discovery for webservice operations following clustering of parameters into semantically mean-

ingful concepts. A set of operations are considered to be similar when they take similar inputs and produce similar outputs and similarity of concepts associated with parameters is based on TF/IDF measure.

4.2 Interface Match Approach

In case of information retrieval approach, words that are common in the collection of documents are examined to assess the similarity. Interface similarity is addressed at different levels [32]. The query is specified as an interface giving expected input, output, and operations. The template for the query is similar to that of service publication. Similarity among interfaces ($Sim_{\text{Interfaces}}$) is computed by assessing similarity between operations ($Sim_{Operation}$) and quantitative score is computed. As each operation consists of inputs and outputs, inputs specified in the query interface (S_q) are compared with inputs of a selected service published in repository (S_r).

$$Sim_{\text{Interfaces}}(S_q, S_r) = \text{Max} \sum_{i=1}^{p} \sum_{j=1}^{q} Sim_{Operation}(O_{1i}, O_{2j}) \times x_{ij}$$

$$x_{ij} = \begin{cases} 1 \; when \; Combining \; O_{1i} \; with \; O_{2j} \\ 0 \qquad\qquad\qquad else \end{cases}$$

Assuming that S_q has p operations and S_r has q operations. Similarity is calculated from every operation of S_q to every other operation in S_r. The comparison of services results in operation matrix with operations from query service and service from repository.

$$Sim_{\text{Operations}}(O_1, O_2) = \text{Max} \sum_{i=1}^{m} \sum_{j=1}^{n} Sim_{Input}(I_{1i}, I_{2j}) \times x_{ij} + \text{Max} \sum_{i=1}^{u} \sum_{j=1}^{v} Sim_{Output}(I_{1j}, I_{2j}) \times y_{ij}$$

Assuming that operation O1 has m input parameters and u output parameters, O2 has n input parameters and v output parameters. To compare inputs, for similarity calculation, parameter name (Sim_{Lexical}) and data types (Sim_{Datatype}) are compared. Parameter names being words their lexical (semantic) similarity is computed.

$$Sim_{\text{Input/output}}(I_1, I_2) = Sim_{\text{Lexical}}(I_1.Name, I_2.Name) + Sim_{\text{Datatype}}(I_1.DT, I_2.DT)$$

Input Sim_{Input} and Output Sim_{output} similarity score is calculated by pairwise correspondence of input–output parameter list. Similarity between operations is score

associated with input/output parameter lists that maximizes the sum total of the matching scores of the input/output individual pairs.

Data type similarity is syntax matching and lexical similarity is semantic matching.

4.2.1 Syntax Matching

In [23], Zaremski and Wing have proposed exact and relaxed signature matching for retrieving functions with matching signatures. Exact match is when the sequence of variables and their data types in query match with sequence of variables of a function allowing the renaming of variables. But no specific mechanism is used to find the similarity between variable names. Relaxed match is partial match that considers generalized and specialized cases of type match, reordering of parameters. For an integer, float or long is generalized case and for Boolean int is specialized case. Also, pre- and post-conditions are used to determine the type of match.

Syntactic and semantic matching approaches [25] have been proposed to assess the similarity between two WSDLs. Services are compared on matching data types and identifiers. For syntax matching if both services have same data types, maximum score is assigned else lesser score is assigned.

A similar syntax matching is proposed in [33] with algorithm to match data types of simple and complex data elements. The paper [34] presents comparative study on using various algorithms, viz., total enumeration, greedy and Kuhn-Munkers for measuring the similarity between data types of service messages associated with each operation.

4.2.2 Semantic Matching

Targeting toward automatic service discovery researchers has introduced semantics into the service descriptions. Ontologies are used in service descriptions to clearly specify what each of the terms is meant by service provider.

Distance Measures

In [25], semantics is used for matching identifiers (names of operation and messages). Semantic matching is powered by wordnet [30] to identify identical words, synonyms, and hierarchical semantic relations (hypernyms, hyponyms). Hypernym is the generalized term and specific instance of it is hyponym. Number of semantic links between words in wordnet hierarchy is number of hierarchical links. Identical words get high score of 10, synonyms get score of 8, and for hierarchical relation between words score is 6/(number of hierarchical links). Similarity score between webservice and query service is the sum of syntax and semantic matching scores between their corresponding WSDLs. Each service in repository is matched against query service and the sum total of matching scores is computed. Based on score services are ranked and top 50% are returned as result.

Table 4 Concept relationship definitions

Relationship	Description
Exact	Request and advertisement are same
Plug-in	Request is sub class of advertisement/ Advertised service has more general concept than request
Subsume	Advertisement is subclass of request/ Advertised service has smaller concept than request
Fail	No relationship exists between request and advertisement

Semantic matchmaking of services based on input and output descriptions of webservices and also precondition and effect is proposed in [35]. Conditions in OWL-S are described using Semantic Web Rule Language (SWRL). Matching of advertised services with that of requested service results in different degrees of matching. The score is calculated based on subsumption-based scoring, semantic distance-based scoring, and wordnet-based scoring.

For subsumption, score calculation input terms of advertisement are compared with input terms of request based on the type of relation in ontology tree, match may fall into one of the categories as shown in Table 4. Accordingly, scores are assigned for results [35, 36].

Semantic distance is calculated by identifying subsume relationship in ontology and is assigned weight between 0 and 1 this is subsumption score. Semantic weight distance between any two concepts x and y is the product of semantic weights on the path from x to y. And this is multiplied by subsumption score.

In [24], service concept model is defined along with signature Service Interface (SI) and it includes Common Properties (CP), Special Properties (SP), and Quality of Service (QoS). CP of webservices are service name, service key, service description, service owner, and service URL and are populated from UDDI registry. Special properties like type of media stream are stated here. Service interface is defined with Service Name (SN) Input and Output parameter (IM/OM) list. SI = <SN, IM, OM> QoS is defined as tuple QoS = <Time, Reliability, Fidelity, Security>. Performance of a service with respect to time is characterized by time to process, time to delay, time to repair, and time to failure. Reliability is a function of failure rate. Fidelity is function of effective design. All QoS values are double and are in the range of 0–1.

4.2.3 Logical Reasoning

In [37], service interface details are specified in OWL-S profile, goal, and mediations are specified in WSMO. Webservice matchmaking algorithm is proposed that extends object-based matching techniques (used in Structural Case-based Reasoning) along with it description logic reasoning over profile instances is used. It explores structural knowledge of ontology other than subsume.

Two profile aware similarity metrics DLH and DLR are proposed. Where Description Logic Hierarchy (DLH) represents similarity of two ontology concepts based on their hierarchical position. Hierarchical filters between two ontology concepts fall into one of the exact, plug-in, subsume, and sibling. Distance between concepts is measure of Edge Count (EC) and Upward Cotopic (UC) measure. EC is number of edges found on the shortest path between two concepts. UC is the ratio of common superclasses of two concepts. DLH corresponds to taxonomical similarity.

Description Logic Role (DLR) denotes the similarity between query and advertised services in terms of functional (IO) and nonfunctional (data types) similarities. Functional similarity is geometric mean of DLH similarity of input and output sets of two profiles. Functional similarity follows three filters to match IO parameters exclusive (x), exclusive input (xi), and exclusive output (xo). Corresponding to query and service having same number (x) of input and output parameters, only input parameters are same (xi), only output parameters are same (xo).

Total score is weighted average of taxonomical, functional, and nonfunctional similarities. Weights are assigned based on user requirement. Along with these profiles, taxonomies are considered as filters in retrieving matching profiles.

In [38], a logical framework is proposed for semantic webservice discovery. The services are specified in WSMO and the components of WSMO, i.e., goals, capabilities, ontologies, and mediators are specified by F-Logic (frame based logic) expressions.

For this proof obligations for service discovery are stated at three levels service discovery, service contracting and the discovery query. A proof obligation is an established logical entailment for a service to be considered as a match for a discovery goal. While service discovery proof obligation is, for a given user goal, the service will be able to satisfy user goal.

Service contracting proof obligation is to check if the actual inputs, outputs, preconditions, and effects are met by service. The discovery query obligation is to find all such services meeting user goal.

To check satisfiability of first obligation after user states goal, mediator constructs input that is appropriate for set of services mediated by this mediator and also converts goal into postcondition expressed in service ontology. For service contracting, first obligation is extended to check precondition and effect. For discovery query transaction logic is used, here the service effects are hypothetically inserted into knowledge base and a query is framed as new hypothetical state if it is true, then query has a match. The assertion is rolled back after all services matching the requester goals are retrieved. FLORA-2 a logical reasoning engine is developed to realize the framework.

In [39], two-phase service discovery is proposed. Here, functional descriptions of services are specified as change of states in abstract state-space model where preconditions form the initial state and post-conditions form the final state. Each service request is mapped to a goal template where the precondition and effect are described in first-order logic, and in second phase, the instances that meet the input are selected from identified set.

In [40], the services are specified in Web Services Modeling Language (WSML) describing goals, ontologies, services, and mediators. Rules specified in WSML are based on F-Logic. Syntactic matching of services is performed on transforming service description into weighted keyword vector and one of the similarity metrics cosine, extended Jaccard, Loss of Information (LOI), and weighted LOI is applied.

5 Conclusion

Changes in the business scenario to be online are supported by advances in technology. Service-oriented architecture through webservices has not only facilitated business to be performed on the web but also to provide new business as per need. Service specification plays an important role when publishing over the web. Evolution of standards from interface description WSDL to machine processable WSMO is comparatively discussed. A well-defined service that is published is easily discoverable. Service search is performed either on the web or on registry. Different search approaches, viz., information retrieval, signature matching, and semantic distance measures are presented.

References

1. OASIS, U. V. (2004). Uddi spec technical committee draft. URL http://uddi.org/pubs/uddi_v3.htm. Organization for the Advancement of Structured Information Standards (OASIS).
2. Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana (2001). Web services description language. URL http://www.w3.org/TR/wsdl.
3. Martin, D., Burstein, M., Hobbs, J. (2004). Owl-s: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/.
4. Joel Farrell, IBM, Holger Lausen, DERI Innsbruck, Semantic Annotations for WSDl and SML schema URL: https://www.w3.org/TR/sawsdl/.
5. Jos de Bruijn, D. F. M. H. U. K. e. a., John Domingue (2005). Web Service Modelling Ontology (WSMO). URL http://www.w3.org/Submission/WSMO/. [29] http://www.wsmo.org/TR/d16/d16.1/v1.0/.
6. Noy, Natalya F., and Deborah L. McGuinness. "Ontology development 101: A guide to creating your first ontology." (2001). URL: http://liris.cnrs.fr/amille/enseignements/Ecole_Centrale/What%20is%20an%20ontology%20and%20why%20we%20need%20it.htm.
7. Kourtesis, D. and I. Paraskakis, Combining sawsdl, owl-dl and uddi for semantically enhanced web service discovery. In The Semantic Web: Research and Applications, volume 5021 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, 2008, 614–628.
8. Wang, H. H., N. Gibbins, T. R. Payne, and D. Redavid (2012). A formal model of the semantic web service ontology (wsmo). Information Systems, 37(1), 33–60. ISSN 0306-4379. URL http://www.sciencedirect.com/science/article/pii/S0306437911001049.
9. de Bruijn, J., H. Lausen, A. Polleres, and D. Fensel, The web service modelling language wsml: An overview. In Y. Sure and J. Domingue (eds.), The Semantic Web: Research and Applications, volume 4011 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2006. ISBN 978-3-540-34544-2, 590–604. URL http://dx.doi.org/10.1007/11762256_43.
10. Supriya Vaddi, Hrushikesha Mohanty, and R. K. Shyamasundar. 2012. Service maps in XML. In *Proceedings of the CUBE International Information Technology Conference* (CUBE '12). ACM, New York, NY, USA, 635–640. DOI: http://dx.doi.org/10.1145/2381716.2381838.

11. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
12. http://www.w3.org/TR/ws-cdl-10/.
13. http://www.ebpml.org/bpml.htm.
14. Fan, J. and S. Kambhampati (2005). A snapshot of public web services. ACM SIGMOD Record, 34(1), 24–32.
15. X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB '04, pages 372–383. VLDB Endowment, 2004.
16. C. Atkinson, P. Bostan, O. Hummel and D. Stoll, "A Practical Approach to Web Service Discovery and Retrieval," *IEEE International Conference on Web Services (ICWS 2007)*, Salt Lake City, UT, 2007, pp. 241–248.
17. webservicex.net URL: http://www.webservicex.net/new/Home/Index.
18. XMethods.NET URL: http://lig-membres.imag.fr/donsez/ujf/GICOM/GICOM_ENS/exemples/webservices/xmethods/www_xmethods_net.htm.
19. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.
20. Yanan Hao, Yanchun Zhang, Jinli Cao, Web services discovery and rank: An information retrieval approach, In Future Generation Computer Systems, Volume 26, Issue 8, 2010, Pages 1053–1062, ISSN 0167-739X, https://doi.org/10.1016/j.future.2010.04.012. (http://www.sciencedirect.com/science/article/pii/S0167739X10000762).
21. Marco Crasso, Alejandro Zunino, Marcelo Campo, Easy web service discovery: A query-by-example approach, Science of Computer Programming, Volume 71, Issue 2, 2008, Pages 144–164, ISSN 0167-6423, http://dx.doi.org/10.1016/j.scico.2008.02.002.
22. Roger Pressman. 2009. *Software Engineering: A Practitioner's Approach* (7 ed.). McGraw-Hill, Inc., New York, NY, USA.
23. Zaremski, A. M. and J. M. Wing (1995). Signature matching: a tool for using software libraries. ACM Transactions on Software Engineering and Methodology (TOSEM), 4(2), 146–170.
24. Jian Wu and Zhaohui Wu, "Similarity-based Web service matchmaking," *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, 2005, pp. 287–294 vol. 1.
25. Wang Y., Stroulia E. (2003) Semantic Structure Matching for Assessing Web-Service Similarity. In: Orlowska M.E., Weerawarana S., Papazoglou M.P., Yang J. (eds) Service-Oriented Computing - ICSOC 2003. ICSOC 2003. Lecture Notes in Computer Science, vol 2910. Springer, Berlin, Heidelberg.
26. A. Gunay and P. Yolum. Structural and semantic similarity metrics for web service matchmaking. In Proceedings of the 8th international conference on E-commerce and web technologies, EC- eb'07, pages 129–138, Berlin, Heidelberg, 2007. Springer-Verlag.
27. A. Wombacher, B. Mahleko, and E. Neuhold. Ipsi-pf: A business process matchmaking engine. In e-Commerce Technology, 2004. CEC 2004. Proceedings. IEEE International Conference on, 137–145, pages 137–145. IEEE, 2004.
28. Huang, Anna. "Similarity measures for text document clustering." *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*. 2008.
29. C. Platzer and S. Dustdar, "A vector space search engine for Web services," *Third European Conference on Web Services (ECOWS'05)*, 2005, pp. 9 https://doi.org/10.1109/ecows.2005.5.
30. WordNet http://www.cogsci.princeton.edu/~wn/.
31. Ruiqiang Guo, Jiajin Le and XiaLing Xia, "Capability Matching of Web Services Based on OWL-S," 16 th International Workshop on Database and Expert Systems Applications (DEXA'05) 2005 pp 653–67.
32. Jian Wu and Zhaohui Wu, "Similarity-based Web service matchmaking," *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, 2005, pp. 287–294 vol. 1.
33. Yiqiao Wang and E. Stroulia, "Flexible interface matching for Web-service discovery," *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, 2003, pp. 147–156.

34. G. Tretola and E. Zimeo, "Structure Matching for Enhancing UDDI Queries Results," *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, Newport Beach, CA, 2007, pp. 21–28.
35. Ayse B. Bener, Volkan Ozadali, and Erdem Savas Ilhan. 2009. Semantic matchmaker with precondition and effect matching using SWRL. *Expert Syst. Appl.* 36, 5 (July 2009), 9371–9377. DOI=http://dx.doi.org/10.1016/j.eswa.2009.01.010.
36. Mehmet Şenvar and Ayşe Bener. 2006. Matchmaking of semantic web services using semantic-distance information. In *Proceedings of the 4th international conference on Advances in Information Systems* (ADVIS'06), Tatyana Yakhno and Erich J. Neuhold (Eds.). Springer-Verlag, Berlin, Heidelberg, 177–186. DOI=http://dx.doi.org/10.1007/11890393_19.
37. G. Meditskos and N. Bassiliades. Structural and role-oriented web service discovery with taxonomies in owl-s. IEEE Transactions on Knowledge and Data Engineering, 22(2):278–290, 2010.
38. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A logical framework for web service discovery. In ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, volume 119. Hiroshima, Japan, 2004.
39. Stollberg, M., U. Keller, H. Lausen, and S. Heymans, Two-phase web service discovery based on rich functional descriptions. In The Semantic Web: Research and Applications. Springer, 2007, 99–113.
40. Klusch, M. and F. Kaufer (2009). Wsmo-mx: A hybrid semantic web service matchmaker. Web Intelligence and Agent Systems, 7(1), 23–42.

Non-functional Properties of a Webservice

N. Parimala and Anu Saini

Abstract A webservice is intended to support communication between various applications over the world wide web. There are three types of properties that are associated with a webservice—functional, behavioural and non-functional. Non-functional properties are defined as the constraints on the functional properties and behavioural description of a webservice. Invariably, if services offer similar functionality then they are differentiated and chosen on the basis of non-functional properties. In this chapter, first, the manner in which non-functional properties are categorized is explained. Next, the specification and discovery of these properties are addressed. Towards this, specification and discovery in a single service as well as in compositions are discussed. Webservice composition can be a static way or dynamic. Both these aspects are dealt with.

Keywords Webservice · Non-functional properties · WSDL · UDDI Webservice composition · BPEL

1 Introduction

Service-Oriented Architecture (SOA) can be defined as an architectural software concept that outlines the use of services to support business requirements. In SOA, resources are made available to other participants in the network as independent services that are accessed in a controlled way [39]. When these services publish their details and interface information on the web, it is termed as 'webservice'. Webservices are platform and language-independent. Webservices are available over the web and they can send and receive data as an XML document. The three core

N. Parimala (✉)
School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi 110067, India
e-mail: dr.parimala.n@gmail.com

A. Saini
G. B. Pant Government Engineering College, Okhla–III, New Delhi 110020, India
e-mail: drsainianu@gmail.com

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_3

XML specifications of webservices are WSDL (Web Service Description Language), UDDI (Universal Description Discovery and Integration) and SOAP (Simple Object Access Protocol).

A service is described using WSDL. It is possible to specify the functions that a service provides using WSDL. But, it is restricted to specifying only the functional properties of a service. However, functional, behavioural and non-functional properties are associated with a service [36]. Non-functional properties are constraints over the first two. When two services are almost the same in terms of the functions they offer, non-functional properties (NFPs) can be used to choose one over the other.

In this chapter, we study the specification and discovery of services based on their NFPs. The layout of the chapter is as follows. Section 2 gives a brief introduction to SOA architecture. The NFPs are described in Sect. 3. The manner in which NFPs can be specified for a single service is dealt with in Sect. 4 followed by the discovery of services based on NFPs, in Sect. 5. NFPs can be associated with composite services as well which is considered in Sect. 6. Section 7 is the concluding section.

2 Architecture of SOA

The architecture of SOA is shown in Fig. 1. It consists of three core XML specifications which are defined below.

WSDL: It defines the XML grammar for describing services as collections of communication endpoints capable of exchanging messages.

UDDI: It is a registry, which is used to store and locate the desired service. In UDDI registry, search can be performed in many ways like search by specific company name, by the name of a particular service, or maybe the different types of service. UDDI can be used by consumers to discover the desired service and by the provider to promote the available services.

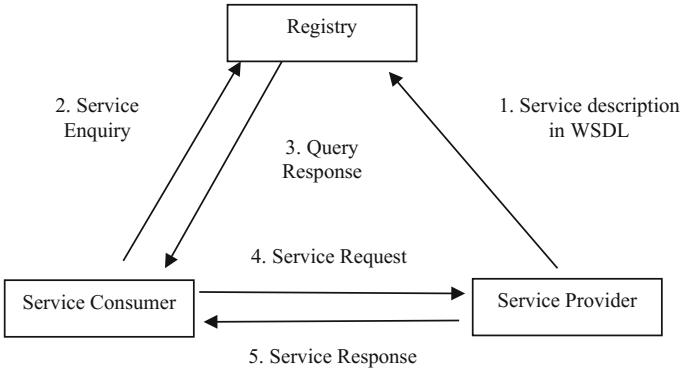


Fig. 1 Service-oriented architecture

SOAP: It is a lightweight protocol for exchange of information [25]. It is based on XML and consists of three parts: a SOAP envelope (describing what is in the message and how to process it); a set of encoding rules; and a convention for representing RPCs (Remote Procedure Calls) and responses.

First, the service description is published by the service provider in the UDDI registry. Then the service consumer requests or receives a contract for some particular service from UDDI registry. Finally, the service provider provides a service that performs some business function at the request of the Service Consumer. All the communication among various entities of SOA is done using SOAP protocol.

According to W3C [6], a webservice can be defined as follows:

> A webservice is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the webservice in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

3 Non-functional Properties

Three different properties are associated with services: (1) functional, (2) behavioural and (3) non-functional [36]. The functions that a service can perform are described by the functional properties. These functions are used by a consumer of the service. The manner in which a user/consumer interacts with the service in order to invoke its functions is described by the behaviour property. Further, in order to provide the functionality to its user, a service may interact with other services. The manner in which this interaction takes place also forms part of the description of the behavioural properties. Finally, constraints over the functional and behavioural properties are specified as non-functional properties.

Functional properties that are used to publish or find webservices are basic information like name, contact information, etc. However, in some situation, it is inadequate to find the service only on the basis of the basic information. Further, with the growing number of services, there are a number of services which have similar or sometimes, even identical, functionalities. There is a need to add more information to the service to distinguish one service from the other. The additional information is available as non-functional property of a service. These properties can be, for example security or performance. As a result, the user can now choose a service based on some non-functional properties as well.

According to [16, 37] a service is described completely only when the non-functional aspects are also described. To discover, select and substitute services, NFPs are very important [48]. NFPs are also necessary for service management, enabling service negotiation, composition and substitution [37].

Recall the architecture of SOA, where a service is first published in the UDDI, then a user discovers the service and finally requests the provider for the service. NFPs are incorporated within SOA in all the stages, namely, specification, publishing

Table 1 Non-functional properties

Availability	Accessibility	Accuracy	Audit trail	Authorization
Best practices	Capacity	Classification	Control	Compliance
Encryption	Execution time	Integrity	Interoperability	Introspection
Latency	Non-repudiation	Penalty	Price	Refresh time
Reliable messaging	Response time	Robustness	Scalability	Security
Throughput	Usability			

and discovery of webservices. Further, composite services can be composed using services supporting NFPs. These, in turn, support NFPs.

Quality of Service (QoS) attributes form a major component of NFPs. An extensive list is given in [26]. Some important ones are shown in Table 1.

Some of the QoS attributes are explained below [33].

Availability quality aspect of a webservice indicates whether it is available for immediate consumption or use. It is the probability of availability of a service. Higher the probability the greater is the chance of finding the service. Smaller probability values indicate that the service may not be available at all times. If a service is unavailable, then it takes a finite time to repair the service. This finite time is referred to as time to repair (TTR). Smaller values of TTR will increase the service availability.

Accessibility attribute refers to the quality feature of a service as to whether it is capable of serving a webservice request. Accessibility and scalability are interrelated. Highly scalable systems can provide high accessibility. A webservice may be available but not accessible.

Accuracy measures the error rate generated by a webservice. Obviously, it is desirable to minimize the number of errors that the service generates over a time period.

Integrity deals with the manner in which unauthorized access to data is prevented and the manner in which the correctness of the interaction is maintained by the service. To a great extent, integrity can be achieved by the use of a transaction. A transaction is a sequence of activities to be treated as atomic.

Performance is measured in terms of time taken by the service to execute. It can be considered in terms of throughput, response time, execution time, transaction time, and latency. Throughput can be defined as the number of service requests that can be managed within a particular time. The time taken to service a webservice request is the response time. The time a consumer waits for a response after making a request is termed as latency. Time taken by a webservice to complete its execution is the execution time. Similarly, the time taken to complete a transaction is the transaction time. High throughput, fast response and transaction time, low execution time and latency values indicate good performance of a webservice.

One aspect of **reliability** deals with whether the delivery of messages between sender and receiver in the specified order has been achieved. The second aspect

deals with the ability to maintain the service and service quality. The third aspect of reliability can be the number of failures per month or year.

Webservices have to follow rules, comply with standards and the established service-level agreement. This quality aspect is referred to as **regulatory**. Standards such as SOAP, UDDI and WSDL are already available. Webservices have to not only adhere to these standards but also the correct versions so that consumers of services invoke the webservices correctly.

A webservice displays **robustness** if it can function correctly even in the presence of invalid, incomplete or conflicting inputs.

Scalability implies that even when the number of requests increases, it is possible to service the requests.

Since webservices are invoked across public internet, **security** aspect must also be considered. There is a need for a secure message exchange between the provider and the consumer of the service. Towards this, confidentiality, encryption and access control must be provided.

3.1 Categories of NFPs

NFPs such as performance, scalability, reliability, availability, stability, cost, completeness, channel, charging styles, settlement, payment, payment obligations, security, trust, etc., are specified as QoS properties in [14, 52].

Some researchers, instead of dealing with individual property, classify the NFPs into categories. In [7], NFPs are divided into two categories: QoS properties and context properties. QoS-based properties are divided into two main categories namely execution and security while the main subcategories of context properties are business and environmental. The former is further divided into execution and response time, accessibility, compliance, possibility of success, availability and security into encryption, authentication and access control. The business properties are further divided into cost, reputation, organizational arrangement, payment method, monitoring and environmental properties into temporal and location properties.

In [32], the authors divide the QoS attributes into two categories: quantitative attributes, e.g. response time, availability, reliability and throughput and qualitative attributes, e.g. security and privacy.

In [26], NFPs are classified according to their reaction in service aggregation, composition, architecture, etc., and according to their semantic description. The first category has five levels of axes, which are domain-dependency, negotiability, type of value, direction and place of measurement. The semantic view has three levels: business, system and service and four categories: performance, dependability, security and trust and cost and payment.

Immaterial of the categorization, NFPs have to be published along with a service. Further, it should be possible to discover the services based on NFPs. Both these aspects are considered in this chapter.

4 Extensions for Specification

By using the standard WSDL definition, it is not possible to incorporate the NFPs as WSDL is designed for expressing only the functional properties. Efforts have been made to extend WSDL to include NFPs in the definition of a webservice like specifying testing factors, handling various versions, security parameters, quality of service, etc. Before explaining the extensions, WSDL is briefly explained first.

4.1 Web Service Definition Language

Webservice can be defined using Web Service Definition Language (WSDL). WSDL is a specification which is based on XML grammar and is used to define and describe the webservices. WSDL also helps the user to access the service. According to W3C, WSDL describes network services as a set of endpoints. They operate on messages. Messages themselves contain either document-oriented or procedure-oriented information [15]. WSDL describes the functionalities of a service. It also describes the manner in which the service can be invoked. Service provider and service consumer can communicate with the help of WSDL. WSDL elements are divided into two parts—abstract and concrete. Abstract part contains various elements: <definition>, <types>, <messages>, <porttype> which further contain <operation>, <inputmessgae>, <outputmessage> as shown in Fig. 2a. Concrete part has <bindings>, <service> further divided into <port> as shown in Fig. 2b.

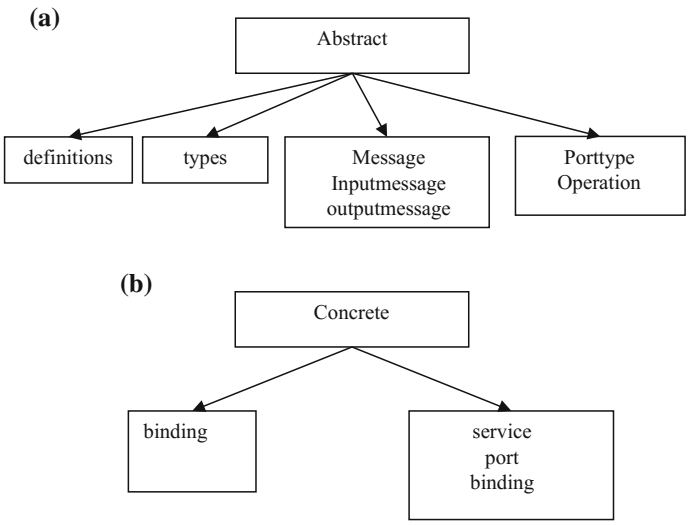


Fig. 2 **a** WSDL abstract elements. **b** WSDL concrete elements

WSDL document consists of the following elements to define a network service [15].

Definitions: The root element of the entire WSDL documents must be the definitions element. It specifies the name of the webservice, declares multiple namespaces utilized all through the rest of the document and comprise every service element explained here.

Data types: The data types describe the different types of data to be used in the messages between the client and the server. It is normally in the form of XML schemas though other mechanisms are also possible.

Message: It is either a whole document or arguments that are to be mapped to a method invocation or it is an abstract definition of the data.

Operation: It is an abstract definition of the function that will receive and process a message.

Port type: It is the collection of operations. It can be mapped to multiple endpoints.

Binding: The concrete protocol and data formats defined for a particular port type for the operations and messages is termed as binding.

Port: The target address of the service communication is provided by port, which combines a binding and a network address.

Service: The service definitions in the file are encompassed by a set of related endpoints called service; binding is mapped to the port by services which also incorporate any extensibility definitions.

Import: For importing other WSDL documents or XML schemas the import element is used.

4.2 Extensions to WSDL

We first consider the specification of NFPs in a single service using WSDL.

In [2], using the extension element of WSDL, NFPs are specified. In this extension, it is possible to describe, measure and update the value of NFPs. A complex type, NFpropertype, is predefined to define the properties that have to be enforced. Below is an example which specifies that the response time for the operation 'subtract' has to be 2 s.

```
  <nf:operation name="subtract">
  <nf:property
name="http://www.ibm.com/wsdl/NFProperties/ResponseTime
"
      dataType="decimal" unit="seconds"
      validUntil="2009-12-31T12:00:00">
      2
  </nf:property>
  </nf:operation>
```

Both WSDL and UDDI are extended in [28] to support *versioning* of a webservice. Here, it is possible to maintain multiple versions of a single as well as multiple service

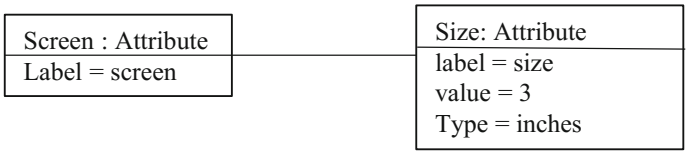


Fig. 3 An example

interfaces. WSDL is extended at both service and operation levels. The version can be specified for all the elements of an operation level. That is, types, messages, interface, bindings and service endpoints can have different versions. If the main elements of a service are changed at the same time, then the version is assigned to the service. That is, if WSDL elements like interface, types, messages and other elements are changed at the same time, then a service-level versioning is used. The version identifier consists of major.minor versions numbers. Below is an example of a service-level versioning, wherein the changes in this specific version are described. Service consumers can programmatically query version information to plan the development activities.

```
  <wsdlx:versions mode = "servicelevel">
  <wsdlx:version  vid  =  "2.0"    type  =  "backward-
incompatible" intent ="revision">
  <wsdlx:previousVersion vid = "1.0"/>
  <wsdlx:versionInfo> "modified XML…."
  </wsdlx:versionInfo>
  </wsdlx:version>
```

In [18], a model-driven architecture has been used to create an accurate mapping between physical object and webservice. The mapping is accomplished by extending WSDL with NFPs. First, a metamodel for WSDL is built. This is expressed in UML notation. There are ten classes. The class Definition is composed of Types, Import, Service, Binding, Message and PortType. The class Service is composed of class Port. Class Message has class Operation and one or more class Part.

Next, this model is extended by defining a two-level descriptive model which has two classes—Attribute and Description. *Class Attribute* gives the overall description of the attribute and the class Description shows the detail. For example, in Fig. 3 the size of the Screen has to be 3 inches.

This model is more suitable for the composed service rather than single service as it is complex if it is extended for all NFPs.

WSDL metamodel is used also in [20]. Here, the *performance* attribute of quality of service such as response time, throughput, utilization is considered. WSDL is extended to incorporate the description of the performance characteristics of a webservice. The extension is called P-WSDL (Performance-enabled WSDL). The specification is a two-step meta data-driven process. In the first step, WSDL metamodel is derived from the WSDL XML Schema as in [18]. In the next step, the WSDL metamodel is extended by applying a metamodel transformation that maps

the elements of a source metamodel (i.e. the WSDL metamodel) to the elements of a target metamodel (i.e. the P-WSDL metamodel). In this model, it is possible to compare the actual performance with the expected performance.

In [1], UDDI and WSDL are extended for expressing *security* of webservices. The authors extend both WSDL and UDDI, by a new element <securityParameters>, for publishing and discovering a webservice. The security parameters are added as optional parameters to provide a secure environment in which transactions can occur. These parameters include provider encryption and signature public keys, conditions for acceptability of user keys for encryption and signature, access control policies and data use policies.

To incorporate QoS information WSDL is extended to WSDL-S [54]. In this study, quality attributes are classified into domain-independent and domain-specific one. Domain independent are attributes like availability, performance, etc.; the domain specific ones pertain to the domain. For example, the QoS attribute frame rate is only in media services. The service provider advertises these QoS attributes by incorporating those into WSDL documents. The service with the QoS attributes is published in UDDI and the related WSDL URL is sent to Quality Manager. A service requestor sends the expected quality descriptions along with function descriptions to Quality Manager. Table 2 is an example of such a request. Quality Manager first selects all services that satisfy the functional properties. Within these, the most suitable service which meets user's expectations of QoS is chosen.

Ontologies have been used to define the semantics of QoS. The upper ontology is built as the basis of quality ontologies. QualityIndependent ontology has domain-independent quality attributes and QualitySpecific ontology has the ontological terms of a particular domain.

NFP is considered as a different property from QoS in [38]. The authors have proposed a new webservice named as 'Criteria Based Webservice', where criteria as a NFP are associated with a webservice. Criteria are extra added properties to the webservice. For example, criteria for 'book flight' service could be that it has 'a window seat'. Therefore, to incorporate the newly added criteria with a webservice the authors have extended WSDL to X-WSDL. To do this, WSDL schema is extended by adding a new keyname 'criteriaservice' as shown in Fig. 4.

An example, which shows the criteria 'Window seat' and 'Vegetarian food' associated with FlightRoomService is shown in Fig. 5.

Table 2 A requestor with quality constraints

$QS(S) >= 0.85$
$E_R(downloadSpeed) = 0.85$
$E_R(ResponseTime) = 0.55$
$booktype = pdf$

```
<xs:element name="definitions">
    <xs:key name="criteriaservice">
        <xs:selector xpath="cr:service"/>
        <xs:field xpath="@name"/>
    </xs:key>
</xs:element>
```

Fig. 4 Schema definition for X-WSDL

```
<wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:cr=" http://localhost:8080/EUDDI/wsdl1"
targetNamespace= "http://localhost:8080/EUDDI/wsdl1.xsd"
<cr:service name=" FlightBookService">
        <cr:criteria name= " Window Seat"/>
        <cr:description name ="    Need window seat"
    </criteriadescription">
        <cr:criteria name= " Vegetarian Food " />
        <cr:description name = "  Vegetarian food should  be available" </criteriadescription">
        <cr:port name=" FlightBook"
                binding="  ">
        </cr:port>
  </cr:service>
</def inition>
```

Fig. 5 Example of the definition of X-WSDL

5 Extensions for Discovery

Having specified NFPs while describing a service, it is necessary to provide a mechanism where NFPs can be specified while discovering a service. The methods range from extensions to UDDI, discovery semantics based using ontology, peer-to-peer systems and agent-based framework. We discuss each of these in turn. Before we do so, the structure of UDDI is briefly explained.

5.1 Universal Description Discovery and Integration

Universal Description Discovery and Integration (UDDI) is a registry for publishing and discovery of webservices. Service providers use the registry to publish their service definitions and the registry is used by the users/requestors to find the required service. These registries aid the businesses to quickly, easily, and dynamically discover webservices and interact with each other. UDDI permits businesses to register their presence on the web [46]. The basic structure of UDDI containing businessEntity, businessService, businessTemplate, tModel is shown in Fig. 6 [10]. UDDI contains

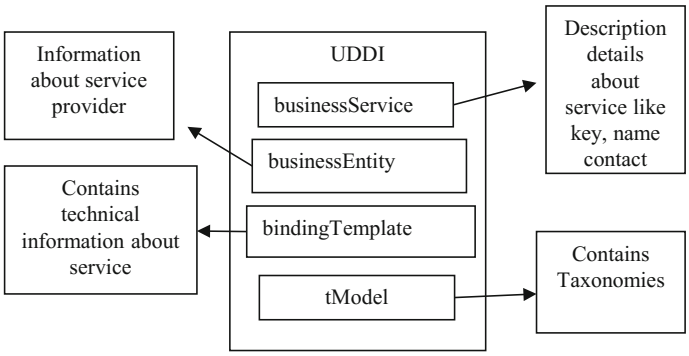


Fig. 6 UDDI structure

three pages—white page, orange page and green page. These pages contain information about webservices. The basic information about businessService is given on white page. The various services offered by businessEntity are given in yellow page. The technical description about webservices is provided on the green page. The three pages are represented in businessEntity, businessService, bindingTemplate and tModel.

businessEntity: businessEntity element includes information about the actual business of a service provider. An element in businessEntity is a Universally Unique ID (key) for individually representing their business.

businessService: businessService provides a list of webservices offered by businessEntity. Each businessService contains a businessEntity key, service key, service name and webservice description.

bindingTemplate: A bindingTemplate is a child of a businessService. Each businessService contains a list of binding templates, which offer information on where to get the service and how to utilize the service. A binding template includes a unique key to specify a businessService to which it belongs and in addition, it contains the access point of the service implementation.

tModel: In UDDI data model, tModels are utilized to specify the technical definitions. Each tModel contains name, explanatory description and one unique key that are referenced by many bindingTemplate.

5.2 Extensions to UDDI

Many extensions to UDDI to incorporate QoS have been proposed. We consider some of them here.

UDDI is extended to UDDIe to include QoS attributes for discovering and describing a service in [43]. There are three main extensions incorporated in UDDIe are given below:

- support for 'leasing' which specifies a limited time period during which the service is available
- support for search on other attributes of a service wherein the businessService class is extended with propertyBag
- extending the find method in UDDI.

UDDIe supports 'Finite' and 'Infinite' leases. The services can be made available by service providers for limited time periods by specifying 'Finite' (for security reasons, for instance)—or may provide the service for an infinite period of time. Service properties are contained in the `propertyBag` entities of a business service. Various attributes of the property are propertyName, propertyType and propertyValue as shown in Fig. 7.

Finally, interaction with the registry system is accomplished by extending three existing APIs. The extensions are to _saveService wherein attributes such as cost of access, performance can be stored using the propertyBag mechanism provided in UDDIe. API _findService is extended to include queries based on Service Property and Service leasing. Lastly, _getService Details API provides the details of the new properties.

UDDIe contains an additional 'blue page'. When a service is published, the QoS information is stored on the blue page. The information stored on the blue page is used by the consumer to discover services based on QoS properties. The G-QoSM framework [3] is used to implement the system. A service request by the consumer is sent to QoS broker. The broker interacts with the UDDIe registry to find the service (with the QoS properties) that satisfies the user's request.

UDDI specifications are extended to offer and utilize the predictions about web-services behaviour in [5]. In this work, a developer is not expected to state the QoS attributes that are supported. A consumer also is not expected to provide QoS attributes that she expects. However, agents are able to pick up the most suitable service at run time. Predictions about a service are calculated and are available for the agents to examine. The QoS properties that are used to predict a service are availability, reliability and completion time. Availability is the probability that the webservice is up; Reliability is the probability that a correct SOAP request will generate the correct SOAP response; Completion Time is arrival the time between arrival and response of the SOAP request. To execute such an extension, a prototype—named

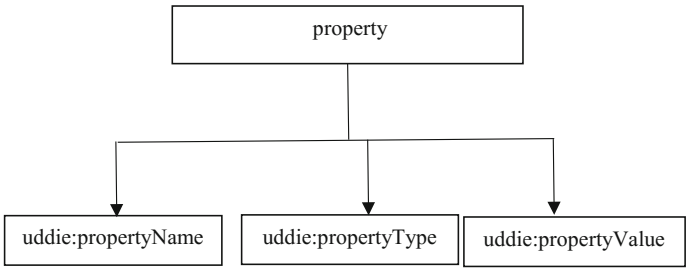


Fig. 7 Property attributes

eUDDIr—has been developed. The bindingTemplate of the UDDI registry has been extended in eUDDIr. The bindingTemplate lists are ordered by the predicted runtime behaviour of the service. Three pairs of qualifiers, for the three attributes of QoS that are supported, have been proposed. The qualifiers for availability are SBPAA and SBPAD where the last letter indicates whether the result set of a qualifier is to be sorted in ascending or descending order. The corresponding tModel names are euddir:sortByAvailabilityAsc and euddir:sortByReliabilityDesc. The qualifiers for completion time are SBPCTD and SBPCTA where, as before, the last letter indicates whether the result set of the qualifier is to be sorted in ascending or descending order. The corresponding tModel names are euddir:sortByCompletionTimeDesc and euddir:sortByPredictedCompletionTimeAsc. For reliability, the pairs are SBPRD, euddir:sortByReliabilityDesc and SBPRA, euddir:sortByReliabilityAsc.

Ran [40] has proposed a webservice registration model which consists of four roles. Webservice supplier, Webservice consumer, Webservice QoS certifier and the new UDDI registry. A provider provides QoS constraints that are supported by the service. These are verified by a QoS certifier. The new UDDI registry is a repository of Webservices with certified QoS. A service consumer can look up the new UDDI registry for a service with the desired functional description and QoS attributes as a constraint.

businessEntity data structure of UDDI is modified to include publisherAssertion data type about QoS attributes that are supported. This data structure is under both the businessService data structure type, and bindingTemplate data structure type, specifies the tModel. UDDI data structure is extended by 'qualityInformation' data type. Below is an example wherein the desired QoS attribute 'availability' to be at least 0.9 (that is available 90% of the time) is embedded in a SOAP request.

```
<qualityInformation>
      <availability> 0.90 </availability>
</qualityInformation>
```

Blum and Carter [11] propose, in a technical note, the different ways in which management information can be stored in UDDI. The overall performance, reliability, availability and throughput are aggregated. tModels have been used to represent the concepts or constructs that are used to describe compliance with a specification. Four alternative methods to store such information in UDDI registries are discussed. In the first method, a reference to an external QoS is defined in the UDDI by introducing the QoS tModel, called QoS information. Each UDDI bindingTemplate contains a QoS Information tModel and adds the QoS Information tModel to the tModel Instance Details collection. In the second method, for each different type of QoS information, new tmodels are created which are subsequently added to the binding templates. The third method is similar to the first method with the difference that it contains the binding template. As a result, the category bag of QoS information tModel has many key references to represent different QoS information. The last method uses the categoryBag in UDDI to store the QoS values. Multiple steps are needed to locate the required service which implies that it may take a long time to locate a service. The system may well turn out to be inefficient.

5.3 Specification Using Semantics

A requester's preferences of QoS attributes in a semantic context is considered in [27]. OWL-Q ontology is extended to capture preferences expressed in a language called QoSPref. Constraints that a service must satisfy are declared as a list of comma-separated Boolean conditions. An example is given below

```
constraints {
      chartType = "time series",
      cost < 10,
      availability > 0.90,
      imageResolution = "720x720",
  responseTime < 8}
```

The main contribution is that a user can specify the preferences among the constraints. For the above example, the user can specify the preferences as

```
preferences {
cost,
availability : high,
responseTime,
  colors : low}
```

The user also states that *high* is to be preferred over *low*. New classes in OWL-Q are introduced. The new classes are QoSDemand class for the specification of constraints and QoSPreference class for a preference rule. The preference rules are used to rank webservices, which may not satisfy the constraints identically.

OWL-S is combined with WSDL to express the semantics of a webservice in [34]. The semantics of a webservice is described in the ontology. Specifically, Service Profile module of OWL-S is used to express functional and non-functional properties of a webservice. Using the semantic definitions, a user is provided with information that helps in searching, discovery, selecting and composing webservices. A service is discovered on the basis of user's requirement. A finer refinement of matching is used in [31]. Here, the match may be specified to be exact, subsumed, plug-in, intersection or disjoint. In these approaches, effort has been made to automatically locate the webservice by using ontologies. Even though considering semantics has major benefits, there are some drawbacks. Ontology definition is a complicated, hard and nonflexible task [29]. Another problem with ontology is their maintenance. An additional difficulty is that it lacks repositories or marketplace where interaction can take place.

QoS concepts of webservices like reliability, execution time, response time, availability, etc., are described by concepts in a QoS ontology and then embedded into service description files [50]. The value of a WoS attribute is normalized with a higher value indicating better compliance. QoS values over a time are collected as feedback when services are actually executed. This data is used by a real-valued time series forecasting technique to predict its future quality conformance. Based on these values the selected services are ranked. To do so, a user query is represented as 'a vector Q of triples $\{q_j, n_j, v_j\}$ where q_j represents for the required QoS attribute, n_j is the level of importance of this quality attribute to the user and v_j is the minimal

delivered QoS value that this user requires' [50, 51]. Simple Additive Weighting method is used to rank services.

5.4 Peer-to-Peer Systems

Webservice can be discovered on the basis of NFPs by using a peer-to-peer system. Here, NFPs can be interoperability, scalability, efficiency, fault tolerance and semantics. These systems are unstructured are decentralized, scalable and self-organizing. Service provider is not needed in peer-to-peer systems for the discovery of service [9]. To provide a service, this system works in a distributed way. Each entity works as a server and client of the peer-to-peer service. In this system, there is no way to determine which peer in the system is more likely to have certain data. Therefore, it leads to an inefficient search [24].

In [24], both the functionality and the behaviour of Webservices are taken into account for discovering a webservice. Once the services which satisfy the functional and behavioural requirements are found, they are ranked on the NFPs. NFPs that are considered are trust and quality ratings. Towards this, a scalable reputation model is built. To start with, a webservice is expressed as a Path Finite Automaton (PFA), a finite automaton with no loops. Two counters, q and t, are associated with each PFA. These represent the quality of the service and the trustworthiness of the service provider, respectively. Users of a Webservice can assign a value for q and t for each webservice and the values are collected over six months. Score for each webservice is computed as $\alpha \cdot t + (1 - \alpha) \cdot q$. The scores are used for ranking the services in the reputation model.

Yu [53] and Vu et al. [51] propose a distributed service discovery framework based on structured peer-to-peer overlays as the service repository network. The providers publish service advertisements with embedded QoS information in P2P-based registries. Any registry peer can be queried by the user for a service with the required functionality and QoS properties. This request is internally routed to the registry peer that can answer it. Once the results are returned to the user, the user can invoke the service. The feedback from the users regarding the QoS properties as experienced by them is taken by the registry peer. A safeguard to distinguish malicious feedback from the genuine ones is built into the system. Communication between registry peers provides the support to the user for locating a service wherever it is, taking feedback and other operations.

5.5 Agent/Broker-Based Systems

Agent framework, referred to as Web Services Agent Framework (WSAF), is proposed in [35]. Service consumers and providers communicate by using an agent framework. First, QoS data is collected from agents which are subsequently aggre-

gated and stored. This data is shared between the consumers and providers. Service providers advertise QoS attributes via WSDL. Service brokers augment UDDI broker registries with agencies with which service agents collaborate. The consumer application can contain agents. The interaction between these is shown in Fig. 8. A service agent queries agencies for a suitable match for the consumer's requirement in terms of functional properties and also QoS preferences or policies.

For a dynamic selection of webservice, a broker-based approach is proposed in [21]. Here, all requirements, functional as well as non-functional are specified. A QoS broker negotiates between the registry and service user. This broker helps the users choose the required service from the registry. A QoS broker consists of three components: *Service Selector*, *Service Publisher* and *Quality* Manager. A service provider uses service Publisher to publish both the functional and QoS properties supported by the service. The QoS broker verifies the QoS properties and certifies them. Upon certification, the service is published in the service registry by the service publisher. Next, is the manner in which a consumer finds a service with the requisite functional and QoS properties. The requirements on QoS properties are expressed as QoS constraints. The constraints can be simple or composite. A simple QoS constraint normally deals with one QoS property. For example, QoS constraint like 'response time should be less than 4' is a simple constraint and is expressed as (RT) < 4. Composite constraints are formed using simple constraints and constraint operators AND and OR. The composite constraints expressed as AND–OR, QC, tree. The edges can be assigned weights to express the different importance of each constraint. The QC tree and the functional requirements are sent to the Service Selector by a service consumer. Service Selector component finds the service that matches these requirements. The important step is the feedback system. The consumer sends the feedback to the quality manager, which updates the QoS properties

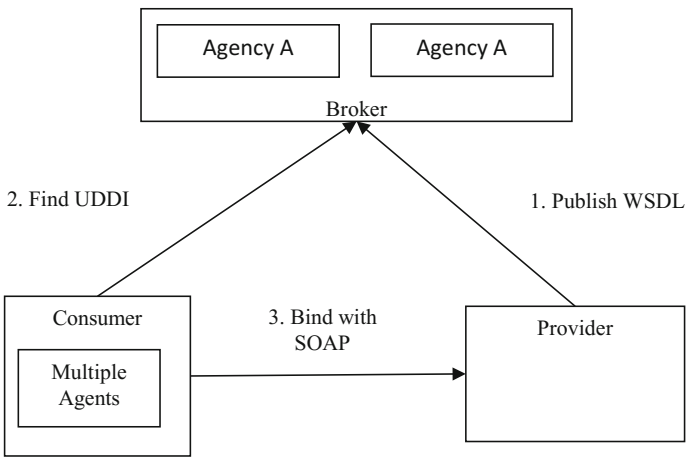


Fig. 8 Agents and agencies in a service-oriented architecture

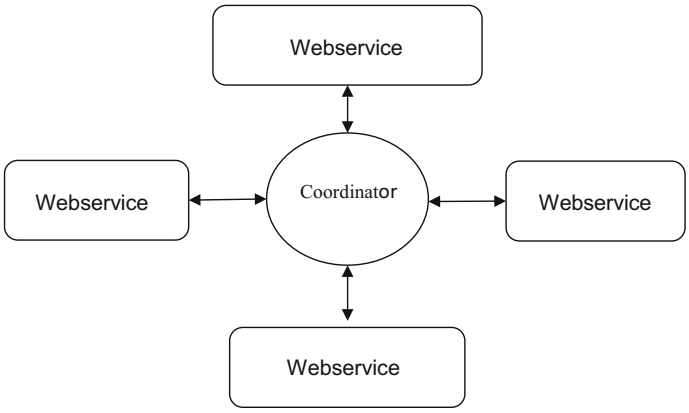


Fig. 9 Orchestration

of the specific webservice. Thus, it is possible to fine-tune the QoS properties with the actual performance of the service.

6 Specifications of NFPs in a Composition

A single webservice which is able to fulfil a user's request on its own is referred to as an atomic service [45]. It does not depend on other webservices to carry out the functionality. Atomic services have limited functionality. When anatomic service is not able to fulfill a user's needs, there is a requirement to compose several individual services. A composition may consist of atomic services as well as other composite services. Services can be combined to form a composition in two different ways—orchestration and choreography [22, 45]. In orchestration, there is a single executable business process. It invokes other services to carry out its functionality as shown in Fig. 9. Orchestration is based on the concept of a central process which interacts with the involved Webservices. It coordinates the activities with the webservices. The webservices themselves do not know that they are a part of the orchestration. The central process coordinates with webservices in such a manner so as to realize the overall goal of the process. The composition describes the sequence and the conditions under which other services are invoked.

On the other hand, in choreography, services interact with each other to carry out a complex task. Choreography represents the interaction between peer services from a global perspective as shown in Fig. 10. The peer services collaborate in a manner so as to achieve the overall goal. Each service is aware of the goal of the business process, its contribution to the business process, the peers with whom it has to exchange messages and the timing of these messages. The services that participate, the rules and order of exchange of messages are described in the choreography.

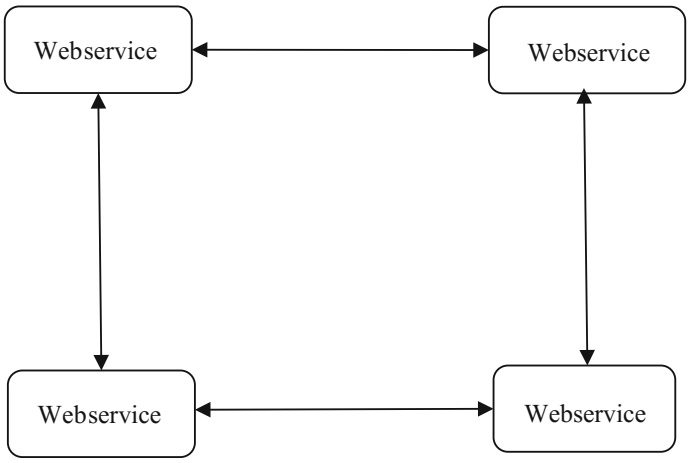


Fig. 10 Choreography

Of the different languages proposed for orchestration, Business Process Execution Language (BPEL) as given in [4] is considered in this chapter.

According to [45], the composition can be either static or dynamic depending on the time when they are composed. In static-based composition, the process model consisting of the tasks is specified at design time. The services that are to be invoked are chosen, bound together as per the process workflow and deployed. In static composition, the webservices are bound early. Now, if the composition is to support the specification of NFPs, then the component webservices must also support NFPs. In other words, the webservices that are chosen must support the specification of NFPs. Further, such services must be available. In the absence of the existence of these services, the composition cannot be taken to completion. Further, if these services are withdrawn at a later time by the developer, then the composition which was running earlier on will fail. The aggregation of the NFPs of the services can be treated as the NFPs that will be satisfied by the composition. In addition, the composition may have its own requirement of NFPs to be satisfied.

On the other hand, in dynamic composition, the webservices can be chosen at run time. Therefore, there must be a support system to discover, select and bind services late, at the time of execution. However, the late binding gives the flexibility to discover webservices which satisfy NFPs at execution time. Undoubtedly, compositions are more likely to go through a webservices environment is highly dynamic in nature.

The composition of webservices using the functional properties can be expressed using BPEL. In this chapter, we explain the manner in which NFPs are incorporated in the composition.

6.1 Static Composition

In static composition, QoS properties are expressed along with the specification of the process model. The webservices which satisfy the QoS properties are chosen at run time.

Quality of Service Language for Business Processes (QoSL4BP), a language which incorporates QoS behavioural logic in orchestration is proposed in [8]. Orchestration Quality of Service (ORQOS), a composition platform, has also been proposed. Policies can be expressed using a policy-based language. The policy consists of three sections: SCOPE, INIT and RULES. QoS settings to be enforced at pre-deployment time for an activity can be defined using the INIT section and at run time using the RULE section. The BPEL activity itself is identified in the 'SCOPE' section. ORQOS platform process has three steps. First, all the services which match the abstract services of the orchestration are picked up provided the total quality of these services matches the SLA (Service-Level Agreement) of the orchestration. Activities for monitoring are introduced in the orchestration. Finally, ORQOS performs QoS adaptation at runtime. Thus, ORQOS guarantees the QoS properties at pre-deployment time and at run time. QoS data is collected both, at the service level and at the composition level. It can be accessed using primitives. Some of these are

(a) REQUIRE which gives QoS information required
(b) PROVIDE which gives QoS information provided by the service
(c) SLAVIOLATION to check if there is SLA violation
(d) SCOPEVIOLATION to check if there is any scope violation
(e) EXCEPTION using FAIL and THROW
(f) PERFORM and PROCESS to check the inbound and outbound SOAP messages.

Transactional and QoS-driven composition approach for composition is proposed in [23]. They combine transactional and QoS requirements in the selection of webservices. The system takes two inputs: a workflow and the user's preferences. Yet Another Workflow Language (YAWL) is used to represent the workflow. The user's preferences are expressed as weights over QoS criteria and as risk levels. The composition manager selects a webservice from the registry. The registry provides ways for not only publishing and locating webservices but also to maintain the metadata for describing the functional, behavioural and NFPs of a web server. The Planner Engine assigns a selected webservice to the concerned activity in the workflow thereby generating an execution plan. This plan is executed enforcing the transactional requirements and the QoS criteria defined by a user.

Performance of a composite webservice is predicted [19]. The work is based on using performance-enabled webservice, P-WSDL given in [20]. Using the composite service specification a class diagram (CD), describing the interface of the composite webservice, and an activity diagram (AD), describing the abstract workflow is produced. When the abstract service is bound to a concrete data, the AD is annotated with performance-oriented data which is available with each concrete service. Different compositions are created. Each composition predicts the performance of the composition. The system can then decide which composition to execute.

QoS properties are added to the BPEL process by using policies in [13]. The approach is based on XPath, WS-Policy and WS-PolicyAttachment. Webservices can publish their policies using WS-Policy [42]. A policy is a collection of policy assertions that can be combined using several operators. The policies can be attached to WSDL documents or any resources using WS-PolicyAttachment [44]. A BPEL process may have its own policies, which are specific to the activities of a BPEL process. Therefore, the authors introduce policy attachment files (*.pat files* for short). The *selector* element is an XPath expression for selecting a set of activities. At deployment time of a BPEL process, webservices which satisfy the policy in the WSDL files are chosen. The policies attached to BPEL and WSDL files are combined for enforcement of the policies. The approach is implemented by extending the Colombo [17] BPEL engine in an event-driven manner so that the policy handling component is notified about different events in the execution of process activities.

For the composition of Criteria-Based Webservice, BPEL is extended to X-BPEL in [41]. Criteria are an additional property which can be associated with a service. To incorporate the newly added criteria information standard BPEL schema needs to be extended. Four new activities get_criteria, accept_criteria, find_dservice and replyX are added. A list of criteria is provided by the user by using get_criteria activity. Here, information flow is from process to user. User responds by using accept_criteria through which BPEL engine accepts the list of criteria. An example of get_criteria is shown below:

```
<xbpel:get_criteria name= " flightservcie"
   Service description =" criteria list for  flight service"
</get_criteria>
```

An example of accept_criteria activity is shown below:

```
<xbpel:accept_criteria name = [window seat, vegetarian food]
      Criteria description= " "
</accept_criteria>
```

To search for the criteria-based webservice from X-UDDI registry, find_dservice has been used. In the result, the registry provides the list of services along with the matched criteria. The replyX activity is used to return the list of alternatives, in the form of an array of strings, to the user.

6.2 Dynamic Composition

As stated before, in dynamic composition, the process model is developed and web-services are chosen at run time. When QoS are specified, the webservices which satisfy the QoS properties are chosen to participate in the composition.

In [30], a framework for the functional composition of webservices is described. A semantic graph is developed which represents the final composition. Initially, Causal Link Matrix (CLM) formalism which represents relevant service compositions is developed. Using this, the set of possible solutions is arrived at. Service and service request contain functional properties such as inputs, outputs, goals, preconditions, etc., and non-functional properties such as cost, security, performance, reliability, etc. These are specified in terms of annotations. These annotations are references to elements defined on ontologies. An example of an annotated service request is:

```
<Non-functional>
<"NFPOnt#Cost" value=10>
</Non-functional>
```

The annotation NFP Ont#Cost says that the cost of the service should be 10. NFPs may be specified with each service. Appropriate functions such as sum, max, min, etc., are used to compute the overall value of the non-functional properties of the composition. For example, if total cost of the composition is to be computed then the function, sum, is appropriate. If a composition does not match the value requested by the users then it is dropped. Finally, a set of valid service compositions which satisfy the NFPs are selected. The value is also used to rank the compositions.

A new platform, Star Web Services Composition Platform (StarWSCoP), is used for dynamic composition of webservice in [47]. StarWSCoP supports specification of QoS properties of a webservice by extending WSDL. BPEL4WS, a composition language is extended to support QoS. Ontology is used to achieve a semantic match for a webservice. To support semantic lookup of a request, UDDI has also been extended. The architecture includes many parts such as an intelligent system to decompose a user's requirement into abstract service. The differences in the webservices are concealed by a wrapper, which acts as a proxy of the webservice. At run time, QoS is gathered by a real-time QoS estimation engine. The engine is fired when the execution of each component of a composite webservice is completed. It is possible that real-time QoS as collected by the engine is at variance with the user's requirement. In such a situation, the engine alters the composite webservice definition.

In [49], a decentralized dataflow model is proposed for webservice composition. Data distribution, reliability, availability and QoS are considered. The framework is made up of nine components: service provider, composer, service requestor, UDDI registry, translator, repository, execution engine, evaluator and matching engine. Data to find a service is increased due to multiple repositories. However, the authors do not show the manner in which the framework can be implemented.

Brahmi and Gammoudi [12] propose an efficient webservice composition approach based on Multi-Agents System (MAS). The completion of a composition fulfils the user's QoS requirements such as speed, cost, reliability, etc. The QoS of the components is aggregated to get the QoS value of the composition. A user requests for a composition fulfils the functionalities as well as the QoS requirements. All similar services are grouped into a Service Class which is managed by a Class Agent. Agents can be connected to each other akin to social networks. These agents

cooperate together to find the optimal and feasible composition. The user specifies the inputs and the required outputs as R = {Inputs = A; Outputs = F}. The composer agent, Compositor, initializes the composition process using the Class Agents. These agents, in turn, use other agents to complete the composition. They send a message to successive agents with their partial composition and QoS value. For example, let the agent A0 choose a service S01 as part of a composition with QoS value 10. A0 sends a message to A2. The message contains {(S01, A0)}#10}. Let agent A2 choose the service S23 as the next part of the composition. The partial composition is indicated by the message ({(S01, A0), (S23, A2)}#40). This says that agents A0 and A2 are triggered and that QoS value so far is 40. The final compositions are returned to Compositor which chooses the composition with the best QoS value.

7 Conclusion

In this chapter, we have dealt with NFPs of a webservice. These are essentially the constraints on the functional and behavioural properties of a service. The NFPs help in choosing a service among those that have similar or even identical functionality. As to what can be specified as a NFP is left to the designer of a webservice. Most of the work, however, has considered QoS as part of NFPs.

NFPs are to be defined with a webservice. WSDL, however, has no provision for specifying NFPs. Typically, extensions to WSDL have been proposed by researchers to incorporate the additional properties. Since WSDL is expressed as an XML document, new elements have been added to define NFPs. The next step is to publish and discover a service with NFPs. Four methods of publishing and discovery are explained in this chapter. The first is by extending UDDI so that a service with NFPs can be published. Further, extensions for searching UDDI based on NFPs are also dealt with. The next three methods, explained in this chapter, do not extend UDDI but build an additional layer to support publishing and discovery of services with NFPs. In the first method, an ontology is built in which NFPs can be expressed and is used for searching for services with NFPs. In the second method, a peer-to-peer system is built over the service discovery framework and in the third, a broker system is developed.

Composition using webservices with NFPs can be static or dynamic. Both the approaches have been discussed. In both the cases, compositions which satisfy the requirements based on NFPs are chosen. In static composition, the services which satisfy the NFP requirement are chosen to complete the composition before executing the process. In dynamic composition, webservices which satisfy the NFP requirement are chosen at run time. These services are used to complete the composition.

References

1. Adams, C., & Boeyen, S. (2002). UDDI and WSDL extensions for Web Service: A Security Framework. *ACM Workshop on XML Security* (pp. 30–35). https://doi.org/10.1145/764792.764798.
2. Agarwal, V., & Jalote, P. (2009). Enabling End-to-End Support for Non-Functional Properties in Web Services. *IEEE Int. Conf. on Service-Oriented Computing and Applications* (pp. 1–8). https://doi.org/10.1109/soca.2009.5410272.
3. Ali, R., RanaO.F., Walker, D., Jha, S., & Sohail, S. (2012). G-QoSM: Grid service discovery using QoS properties. *Computing and Informatics, 21(4)*, 363–382.
4. Alves, A. (2007). *Web Services Business Process Execution Language Version 2.0, OASIS Standard.* Retrieved from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
5. Ambrosi, E. (2005). Extending the UDDI API for service instance ranking. *ISWS*, (pp. 104–110).
6. Austin, D. (2002, 11 14). *Web Services Architecture Requirements*. Retrieved from https://www.w3.org: https://www.w3.org/TR/2002/WD-wsa-reqs-20021114.
7. Badr, Y. (2008). Enhancing Web Service Selection by User Preferences of Non-Functional Features. *4th IEEE Int. Conf. on Next Generation Web Services Practices* (pp. 60–65). https://doi.org/10.1109/nwesp.2008.39.
8. Baligand, F., Rivierre, N., & Ledoux, T. (2007). A declarative approach for qos-aware web service compositions. *In International Conference on Service-Oriented Computing* (pp. 422–428). Springer, Berlin, Heidelberg.
9. Banaei-Kashani, F., Chen, C., & Shahabi, C. (2004). Wspds: Web services peer-to-peer discovery service. (pp.). *In Proceedings of the International Conference on Internet Computing*, (pp. 733–743).
10. Bellwood, T. (2002). *UDDI Version 2.03 data structure reference.* Retrieved from http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.pdf.
11. Blum, A., & Carter, F. (n.d.). *Web Services Management Information - Oasis.* Retrieved February 13, 2017, from https://www.oasis-open.org/committees/download.php/…/UDDI%20WSM-Info-1v7.d….
12. Brahmi, Z., & Gammoudi, M. (n.d.). QoS-aware Automatic Web Service Composition based on Cooperative Agents. *In Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on* (pp. 27–32). IEEE.
13. Charfi, A., Khalaf, R., & Mukhi, N. (2007). QoS-aware web service compositions using non-intrusive policy attachment to BPEL. *International Conference on Service-Oriented Computing* (pp. 582–593). Springer, Berlin, Heidelberg.
14. Chen, Y.-p. (n.d.). Study on QoS Driven Web Services Composition,". *8th Intl. Conf. of APWeb, LNCS*, (pp. 702–707).
15. Christensen, E. (2001). *Web services description language (WSDL) 1.1.* Retrieved from https://www.w3.org: https://www.w3.org/TR/wsdl.
16. Chung, L. (1991). Representation and utilization of non-functional requirements for information system design. *International Conference on Advanced Information Systems Engineering* (pp. 5–30). Springer, Berlin, Heidelberg.
17. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2005). Colombo: Lightweight middleware for service-oriented computing. *IBM Systems Journal, 44(4)*, 799–820.
18. Dai, C., & Wang, Z. (2010). A Flexible Extension of WSDL to Describe Non-Functional Attributes. *2nd IEEE Int. Conf. on e-Business and Information System Security* (pp. 1–4). https://doi.org/10.1109/ebiss.2010.5473641.
19. D'Ambrogio, A., & Bocciarelli, P. (2007). A model-driven approach to describe and predict the performance of composite services. *Proceedings of the 6th international workshop on Software and performance* (pp. 78–89). ACM.

20. D'Ambrogio, C. (2006). A Model-Driven WSDL Extension for Describing the QoS of Web Services. *6th IEEE Int. Conf. on Web Services* (pp. 789–796). https://doi.org/10.1109/icws.2006.10.
21. D'Mello, D., Ananthanarayan, V., & Thiilagam, S. (2008). A qos broker based architecture for dynamic web service selection. *In Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on. IEEE*, (pp. 101–106).
22. Dustdar, S., & Schreiner, W. (2005). A survey on web services composition. *International journal of web and grid services, 1(1)*, I 1–30.
23. El Hadad, J., Manouvrier, M., & Rukoz, M. (2010). TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition. *IEEE Transactions on Services Computing, 3(1)*, 73–85.
24. Emekci, F. (2004). Emekci, F., Sahin, O. D., Agrawal, D., & El Abbadi, A. (2004, July). A peer-to-peer framework for web service discovery with ranking. *Proceedings IEEE International Conference on Web Services* (pp. 192–199). IEEE.
25. Gudgin, M. (2003). *SOAP Version 1.2.W3C recommendation, 24*. Retrieved from https://www.w3.org: https://www.w3.org/TR/2002/WD-soap12-part1-20020626/.
26. Hernandez, E. (2010). *Evaluation Framework for Quality of Service in Web Services: Implementation in a Pervasive Environment, Master Thesis.* LIRIS, INSA, Lyon. https://www.ibm.com/developerworks/library/ws-quality/. (n.d.). Retrieved 02 10, 2017, from https://www.ibm.com/.
27. Iordache, R., & Moldoveanu, F. (2014). QoS-aware web service semantic selection based on preferences. *Procedia Engineering, 69*, 1152–1161.
28. Juric, M. (2009). WSDL and UDDI Extensions for Version Support in Web Services. *Journal of Systems and Software, vol. 82, no. 8*, 1326–1343.
29. Konstanty, H., Kaczmarek, M., & Zyskowski, D. (2008). Semantic web services applications–a reality check. *Wirtschaftsinformatik 50.1*, 39–46.
30. Lécué, F., Silva, E., & Pires, L. (2008). A framework for dynamic web services composition. *Emerging Web Services Technology, Volume II*, 59–75.
31. Li, L., & Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce, 8(4)*, 39–60.
32. Mabrouk, N. (2009). QoS-Aware Service Composition in Dynamic Service Oriented Environments. *ACM/IFIP/USENIX Int. Conf. on Middleware* (pp. 123–142). Springer-Verlag, https://doi.org/10.1007/978-3-642-10445-9_7.
33. Mani, A., & Nagarajan, A. (2010). *Understanding quality of service for web services.* Retrieved from www.ibm.com: https://www.ibm.com/developerworks/library/ws-quality.
34. Martin, D. (2004). Bringing semantics to web services: The OWL-S approach. *In International Workshop on Semantic Web Services and Web Process Composition* (pp. 26–42). Springer, Berlin, Heidelberg.
35. Maxmilien, E., & Singh, M. (2004). Maximilien, E. M., & Singh, M. P. A framework and ontology for dynamic web services selection. *IEEE Internet Computing, 8(5)*, 84–93.
36. O'Sullivan, J. J. (2006). *Towards a precise understanding of service properties.* (Doctoral dissertation, Queensland University of Technology).
37. O'Sullivan, J., Edmond, D., & ter Hofstede, A. (2002). 6. Justin O'Sullivan, David Edmond and Arthur Ter Hofstede. "What's in a Service?," pp. 117–133, 2002. *Distributed and Parallel Databases, vol. 12, no. 2–3*, 117–133.
38. Parimala, N., & Saini, A. (2011). Web service with criteria: Extending WSDL. *In Digital Information Management (ICDIM), 2011 Sixth International Conference on* (pp. 205–210). IEEE.
39. Petritsch, H. (2006). *Service-Oriented Architecture (SOA) vs. Component Based Architecture.* Vienna University of Technology White Paper.
40. Ran, S. (2003). A model for web services discovery with QoS. *ACM Sigecom exchanges, 4(1)*, 1–10.
41. Saini, A., & Parimala, N. (2016). An extension to BPEL for criteria-based web service composition. *International Journal of Computational Science and Engineering, 13(1)*, 87–97.

42. Schlimmer, J. (2004). *Web services policy framework (WS-Policy).*
43. ShaikhAli, A. (2003). Uddie: An extended registry for web services. *Applications and the Internet Workshops. In IEEE/IPSJ International Symposium on,* (pp. 85–89). IEEE Computer Society.
44. Sharp, C. (2004). *Web services policy attachment (WS-PolicyAttachment). Specification.*
45. Sheng, Q. (2014). Web services composition: A decade's overview. *Information Sciences, 280*, 218–238.
46. Srinivasan, N., Paolucci, M., & Sycara, K. (2006). Semantic web service discovery in the OWL-S IDE. *In System Sciences. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on (vol. 6)* (pp. 109b–109b). IEEE.
47. Sun, H. (2003). Research and implementation of dynamic web services composition. *International Workshop on Advanced Parallel Processing Technologies* (pp. 457–466). Springer, Berlin, Heidelberg.
48. Toma, I., & Foxvog, D. (2006). *Non-functional properties in web services.* Retrieved from http://www.academia.edu: http://www.academia.edu/475416/Non-functional_properties_in_web_services.
49. Vadivelou, G., & Ilavrasan, E. (2011). Solution to dynamic web service composition related to QoS. *Electronics Computer Technology (ICECT), 2011 3rd International Conference on (Vol. 5)* (pp. 351–355). IEEE.
50. Vu, L.-H., Hauswirth, M., & Aberer, K. (2005). QoS-based service selection and ranking with trust and reputation management. *In OTM Confederated International Conferences* "*On the Move to Meaningful Internet Systems*" (pp. 466–483). Springer, Berlin.
51. Vu, L.-H., Hauswirth, M., & Aberer, K. (2005). Towards p2p-based semantic web service discovery with QoS support. *International Conference on Business Process Management* (pp. 18–31). Berlin, Heidelberg: Springer.
52. Yu, H., & Reiff-Marganiec, S. (2008). Non-Functional Property Based Service Selection: A Survey and Classification of Approaches. *CEUR Workshop Proceedings, Sun SITE Central Europe.*
53. Yu, L. (2006). Towards P2p-Based Semantic Web Service Discovery with QoS Support. *Workshop on Business Process Management, LNCS,* (pp. 18–31). Springer.
54. Zhu, K. (2008). Quality of Service in Web Services Discovery. *IEEE Symp. of Advanced Management of Information for Globalized Enterprises*, (pp. 1–5).

Service Composition

H. N. Lakshmi and Hrushikesha Mohanty

Abstract As growing number of services are being available, selecting the most relevant webservice fulfilling the requirements of a user query is indeed challenging. Service discovery (or service search) is a process of searching webservices matching a given set of user functional and nonfunctional requirements. Service composition can be defined as creating a composite service, obtained by combining available webservices. In this chapter, we review the various approaches proposed for webservice composition, emphasizing input/output parameter based approaches. Popular algorithms and their implementations are also discussed.

1 Introduction

Service-oriented computing (SOC) has emerged as an important computing paradigm and redefined the way software applications are designed, delivered, and consumed. Services are the fundamental elements used in SOC, to support rapid, low-cost development of distributed applications in heterogeneous environments. Webservices technology is the most promising choice to implement service-oriented architecture (SOA) and its objectives. Nowadays, enterprises are exposing their internal business processes as services utilizing the technology of webservices, thus making them accessible via the Internet.

Services are the basic blocks and service-oriented applications are realized by interoperations among them. As growing number of services are being available, selecting the most relevant webservice fulfilling the requirements of a user query is indeed challenging. Service discovery (or service search) is a process of searching webservices matching a given set of user functional and nonfunctional requirements. Various service search techniques have been discussed in the previous chapter.

H. N. Lakshmi (✉) · H. Mohanty
University of Hyderabad, Hyderabad, India
e-mail: hnlakshmi@gmail.com

H. Mohanty
e-mail: hmcs_hcu@yahoo.com

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_4

Service composition can be defined as creating a composite service, obtained by combining available webservices. If no single webservice can satisfy the functionality required by the user, there is a need to combine existing services together in order to fulfill the user request, resulting in a service composition. Webservice composition still is a highly complex task due to the dramatical increase in the number of services available during the recent years and also due to the dynamic nature of the services.

In this chapter, we provide an overview of the various approaches proposed for service composition, with an emphasis on input/output parameter based service composition. We first provide an overview of service composition in Sect. 2. This section is followed by Sect. 3 that discusses the approaches proposed for input/output parameter based service composition approaches with an emphasis on graphs based techniques. A detailed discussion on relational models and object relational models for service composition is given in Sects. 4 and 5. The conclusion is given in Sect. 6.

2 Overview of Service Composition

Service composition can be defined as the process of aggregating multiple webservices into a single service in order to perform more complex functions. The individual services required for composition are obtained by applying any of the search techniques explained in the previous chapter. Webservices composition techniques can be categorized based on various criteria.

A composed service is a sequence of services that make the composition. Such a sequence can be described in two ways: service orchestration and service choreography. Service orchestration is the coordination and arrangement of multiple services that are often exposed as a single executable business service, as shown in Fig. 1. The business logic and the flow of execution of the various services are predetermined and are under the control of a single endpoint. Transactions between the services involved in the composition, necessary error handling, and the overall process management are done by orchestration. The standard for webservices orchestration is WS-BPEL (or BPEL in short), which is largely supported by the industry.

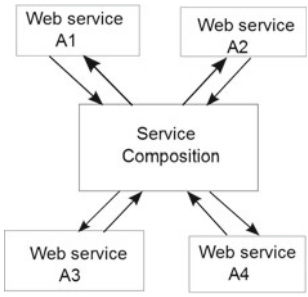


Fig. 1 Service orchestration

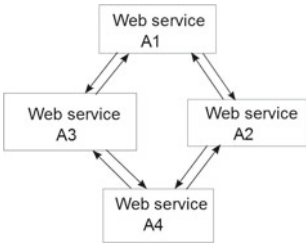


Fig. 2 Service choreography

Choreography employs a more collaborative and decentralized approach to service composition. It allows each of the individual services participating in the composition to describe its part in the interaction. Service choreography is a global description of the services involved in the composition defined by the messages exchanged, interaction rules and agreements between the various endpoints. Interactions among the services involved in choreography are as shown in Fig. 2. The choreography mechanism is supported by the standard WS-CDL (Web Services Choreography Description Language).

2.1 Steps in a Service Composition

From the example depicted above, we can list out the necessary steps in generating a webservice composition as below,

1. Requirement specification.
2. Service selection.
3. Designing service composition.

Requirement specification: In this step, the user specifies the service requirements, in terms of inputs provided by the user, expected outputs and a set of QoS attribute values (optional) that the user expects from the webservice. The requirement is then decomposed, into an abstract composite service, which specifies a set of activities, the control and data flow among them and the Quality of Service (QoS) requirements.

Service selection: For each activity identified in the abstract composite service, suitable services that match the requirements of various activities are searched from either a service registry or a service portal.

It is likely that more than one candidate service will meet the requirements, hence services that best match the user requirements are selected to be composed.

Designing service composition: On selecting all the required webservices required composition, the services are bound to the corresponding activities, thus generating a composite webservice. The constructed composite service is deployed to allow its instantiation and invocation by end users. The result of this step is an executable composite service.

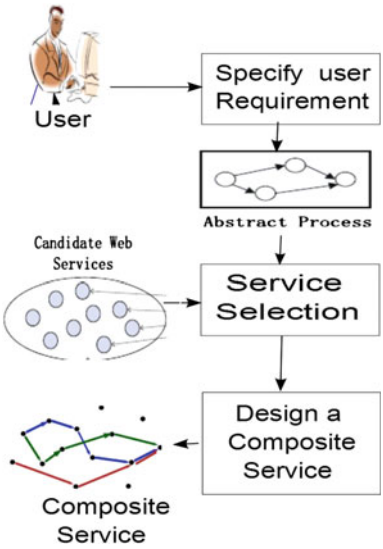


Fig. 3 Steps in service composition

Figure 3 depicts the various steps in service composition process.

2.2 Static Versus Dynamic Composition

Based on the time when webservices are composed, service composition can be categorized as static and dynamic composition [1]. The composition can be done statically where the user builds an abstract model of the tasks that should be carried out during the execution of the webservice. The model is to be finalized before the composition planning starts. Webservices matching of each task is found and one of them is selected for execution of the composed webservice. These approaches are usually implemented through graphs.

This approach works fine as long as the webservice environment, i.e., service providers, and service components do not, or only rarely, change. As and when the old services are replaced by new ones by the providers, inconsistencies might be caused.

Hence, in such a scenario it becomes unavoidable to change the service composition and bind to other services or, in the worst case, even change the process definition and redesign the system. In addition, when there are multiple service providers who often update their services, the static composition may be too restrictive and may not be adaptive to unpredictable updates.

The service environment is a highly flexible and dynamic in that new services may be added by providers on a daily basis and also the number of service providers constantly increasing. Ideally, the service composition process must be able to adapt

to such environmental changes, and also to varying customer requirements, with minimal user intervention. To realize this, the dynamic composition is achieved by creating an abstract model of tasks and selecting individual webservices without much interference of the user in the composition process. Such a service composition requires supporting systems like automatic service discovery, selection, and binding of selected services. However, dynamic service composition is a very challenging task and needs to consider a number of important issues correctness of the composition, time constraints, and transactional support and so on. This type of composition is usually performed using work flow based composition techniques.

2.3 Manual, Semi-automated, and Automated Composition

Webservices composition techniques can be categorized based on the degree of user intervention in the composition process as manual, semi-automated or automatic [1]. Manual composition done by human composers is a mature technique and has been applied universally in the industry. This method can be done either using top-down or bottom-up design approach. In the bottom-up approach, the potential webservices participating in the composition are identified and then connected to each other using a business logic. The top-down approach starts from a specified work flow containing abstract tasks, which are then substituted by concrete webservices, one for each task. Since manual composition relatively demands user intervention and higher costs, the automatic composition is becoming more popular and hence research on automatic webservice composition has become popular, intending to eliminate the user intervention completely.

Automated services composition techniques aim to define a service composition from the specified requirements without any user intervention. Automated services composition approaches typically exploit semantics and artificial intelligence (AI) planning techniques. On receiving a specified requirement from the user, the component services are searched for and best matching ones selected and a composite service specification is generated automatically. Various approaches for automatic selection of webservices are available based on semantic similarity, QoS requirements, etc. However, realizing a fully automated services composition is far from realization. The basic weakness of most of research efforts proposed so far is that webservices do not share common semantics, affecting the automatic selection of services.

A technique that is not purely manual and automatic is simple and practical to realize. This thought gave rise to semi-automated composition, where the process is very much similar to automated composition except that user intervention is allowed at points where selection becomes tough. Such methods aim to assist the user at different steps of services composition process, as and when required.

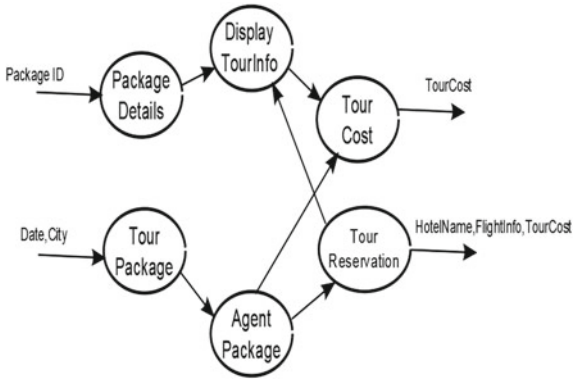


Fig. 4 Parameter-based composition example

2.4 Composition as a Planning Problem

When considering service composition as a planning problem, often webservices are described by the Inputs they receive, Outputs they produce, Preconditions applicable and Effects (IOPEs). The nonfunctional properties of services are often given by QoS (Quality of Service) attributes. Approaches that treat composition as a planning problem start from inputs and preconditions and finally lead to a goal state by going through a sequence of actions [2]. However, most of these proposals have some drawbacks: high complexity, high computational cost, and inability to maximize the parallel execution of webservices.

3 Input/Output Parameter Based Service Composition

There are various approaches proposed for service composition. In this section, we restrict our discussion to approaches that are based on input/output parameters matching of services. A Webservice, ws, has typically two sets of parameters, as a set of inputs ws^I and set of outputs ws^O. Conventionally two services ws_i and ws_j are said to be composable iff $ws_i^O = ws_j^I$, i.e., ws_j receives all the required inputs from outputs ws_i has.

Figure 4 gives an example of input/output parameter based service composition. In the example, when a user queries for a webservice that takes *{PackageID}* as input and provides *{HotelName; FlightInfo; TourCost}* as output, then a composition of services *PackageDetails, DisplayTourInfo* and *TourCost* satisfies the query. Similarly a query that takes *{Date; City}* as input and expects *{TourCost}* as output can be satisfied by two compositions: *TourPackage, AgentPackage, TourCost* and *TourPackage, AgentPackage, TourReservation*.

Table 1 Size of the graph generated for WSC data set

Dataset	No. of edges	No. of paths
Composition-20-32	11,024	930,394
Composition-50-32	58,270	29,363,370

3.1 Graph-Based Service Composition Techniques

A great deal of research [3–6] has been done in recent years on graph-based approaches for webservices composition that consider input/output parameter match of the webservices. Webservices are treated as vertices, edges encode whether one of a service's outputs may serve as another service's inputs. Edges may be weighted based on the matching levels of associated input and output of services.

This transforms service composition search into a graph search problem, i.e. finding paths between vertices in the graph. Many graph algorithms like DFS, BFS, Bellman–Ford, etc., have been applied to solve the problem of service composition.

To understand the complexity of the service composition problem we analyze the size of the graph that is generated using the WSC dataset [7]. WS-Challenge (WSC) data set is a benchmark dataset which consists of webservices with a complex structure. Each webservice in WSC data sets has multiple input parameters and output parameters. The dataset "composition-50-32" means as follows: 50 denotes the number of webservices composition and 32 means that a webservice has 32–36 input and output parameters. The number of webservices in the "Composition-20-32" and "Composition-50-32" are 1000. Table 1 shows the number of edges and the number of paths generated by the WSC dataset.

It can be seen from Table 1 that the number of paths generated for "Composition-50-32" dataset is more than 29 million, showing the complexity of the graph involved in a service registry of just 1000 Webservices. We present few graph-based approaches proposed for service composition.

Precomputed service graphs: Webservices in a repository are visualized as a dependency graph, that capture dependencies among the input and output parameters of the services. The graph is pre-computed using the services listed in the repository where nodes of the graph represent individual services with their input and output parameters and the edges represent input–output dependencies among these services. The various approaches proposed differ in the algorithms used for composition search.

Hashemian et al. [5] propose an approach wherein the dependency graph captures only one-to-one input–output dependencies. This is a restriction since in practice very few services have a single input and/or return a single output. Further, they use a graph search algorithm to search for matching service compositions. Graph exploration algorithms and chaining algorithms are utilized to find a solution for the composition [6]. The proposed approach utilizes backward chaining and depth-first search algorithms to find sub-graphs that contain services to accomplish the requested task. At the end of the network exploration, several composition plans can

be found which are further categorized as: simple composition, serial composition, independent parallel composition, and dependent parallel composition.

Dynamic construction of service graphs: Webservices that may be necessary for a given user requirement are searched from either a service repository or a service search engine and a graph from this set of services is constructed at run time. This reduces the size of the graph generated by a great extent and hence such approaches are faster than the precomputed versions. Also, these approaches support the dynamic nature of service environment where services are updated often by the service providers.

Gekas et al. [4] propose an approach where the search space consists of all the potential webservices that can be part of a work flow. The service registry is viewed as a hyperlinked graph network consisting of services linking to other services. Heuristics regarding the connectivity structure of the repository and how tightly various types of webservices are linked together are derived from the service repository, which is further utilized to select services for service compositions. The composition is done using a recursive depth-first algorithm, which starts from the initial state and tries to reach the goal state following the shortest route possible.

Pablo et al. [2] present an A* algorithm for matching semantic input–output message structure for webservice composition. A service dependency graph is dynamically generated for a given user request from services in a repository and a minimal composition satisfying the request is found using A* search algorithm. Arpinar et al. [3] propose an ontology-driven webservices composition platform. They present an approach which uses weighted graphs for webservice composition. The composition technique aims to find an optimal composition of services considering QoS and semantic matching of parameters. They propose a modified Bellman–Ford shortest-path dynamic programming algorithm to find the shortest sequence from the initial stage at node SI (a webservice in the graph) to the termination node SF (a webservice in the graph).

Most of the graph-based approaches for Service composition create the graph at the time of composition which incurs substantial overhead and use in-memory algorithms for Webservices composition search. The scalability of in-memory approach is limited by the size of physical memory. That makes this kind of algorithms non-scalable. Hence, research on using Relational Database as a scalable repository for storing webservices has become popular recently. The next section describes how services can be stored in a database and discusses the various approaches proposed so far.

4 RDBMS-Based Service Composition Techniques

An overview of how webservices are registered in UDDI and how services are described using WSDL is given in the previous chapters. The WSDL document of a service describes instances of the service using a WSDL service element. Each

Table 2 Example webservices

Service no.	Service name	Input parameters	Output parameters
ws1	HotelBooking	Period, City	HotelName, HotelCost
ws2	AirlineReservation	Date, City	FlightInfo, FlightCost
ws3	TaxiInfo	Date, City	CarType, TaxiCost
ws4	DisplayTourInfo	HotelName, FlightInfo, CarType	TourInfo
ws5	TaxiReservation	CarType, Date, City	TaxiCost

service element in a WSDL document is used to publish a UDDI businessService. The service interface described using WSDL portType and binding is published as a UDDI tModel before publishing a businessService. The various elements of WSDL are mapped to UDDI data structures as follows:

- WSDL portType element is mapped to UDDI tModel.
- WSDL binding element is mapped to UDDI tModel.
- WSDL port element is mapped to UDDI bindingTemplate.
- WSDL service element is mapped to UDDI businessService.

Since current UDDI implementations store the UDDI data structures in a relational database, a first thought would be to include the service parameters in a relational database. This has motivated many researchers to utilize techniques in relational database to solve the service composition problem.

Each operation in a service has an input and output message, each of which in turn may have one or more parameters. For example, a *HotelBooking* service may take *{Period, City}* as input parameters and provide *{HotelName, HotelCost}* as output parameters. A normalized relational database solution to this requires that we store input and output parameters of each operation across multiple rows. Let us consider a simple example to explain it further—Table 2 lists a few example webservices. A relational database solution to this is to assign each parameter a unique parameter id (as in Pars Table) and list the various input and output parameters of webservices across multiple rows as shown in Table 3.

Normalizing the tables in 3 further, we get tables as shown in 4.

We review approaches that utilize relational schemas described above for service composition. Although these approaches use similar relational schemas, they differ in the service details stored, algorithms defined for searching service compositions.

4.1 Relational Databases for Service Composition

Lee et al. [8, 9] were the first to propose utilization of relational database for computing webservices composition. The proposed method extracts information on service

Table 3 Relational schema for webservices

(a) Pars table	
Par ID	Par name
1	Period
2	City
3	Date
4	HotelName
5	FlightInfo
6	CarType
7	TourInfo
8	TaxiCost
9	HotelCost
10	FlightCost
(b) WSInput table	
WS ID	InParsId
ws1	1
ws1	2
ws2	2
ws2	3
ws3	2
ws3	3
ws4	4
ws4	5
ws4	6
ws5	2
ws5	3
ws5	6
(c) WSOutput table	
WS ID	OutParsId
ws1	4
ws1	9
ws2	5
ws2	10
ws3	6
ws3	8
ws4	7
ws5	8

input and output from WSDL of the service and then stores them as tuples in corresponding tables. The *WS table* stores each webservice name along with a unique identification number assigned to it. The names of parameters for webservices are stored in *Pars table*. These parameters may be used as input or output parameters for webservices. The *Input table* and the *Output table* store the input parameters and output parameters of webservices, respectively.

The *Edge table* stores edges are links present in the webservice composition graph. The join operations of *Pars, Input, Output,* and *Edge tables* produce the weighted, directed graph representation of webservices. The approach precomputes all possible webservice compositions and stores in a *Path table*. Intermediate vertices (webservices) in a path are stored in *VisitedWS table*.

On precomputing all possible webservices composition, one can perform webservice compositions search against a given user query. The service search algorithm is described in Algorithm 1.

Algorithm 1: Service Composition Search Algorithm 1

Input: Q:user query
Output: FinalWSs: pairs of webservices

1 Finds all paths (PathIDs) which can be obtained from user inputs and outputs
2 Find all input parameters of services in paths that are not having values
3 **if** Number of input parameters not having values ! = 0 **then**
4 Find new paths that can be obtained from user inputs and outputs
5 Go to step 2
6 **else**
7 FinalWSs are set of all visited webservices in the paths (PathIDs) found

The limitation of the approach discussed above is that it is restricted to services having only single input and output parameter. C. Zheng et al. [10] extended the above approach to handle services with multiple input/output parameters. Also, they consider the semantic similarity of service parameters, using WordNet, to find matching relationships between services. Relationships between the services are stored in a One-way Matching Table (OMT). OMT visualizes a weighted directed graph where each node denotes a webservice and each edge denotes the semantic matching degree between two webservices. Thus, the Service composition problem is simplified to find all reachable paths of two nodes in the graph. They propose a Fast-EP algorithm, based on the algorithm proposed by Lee et al. [8, 9], that improvises the time taken by multiple joins of the table to find all possible service compositions.

Jing Li et al. [11] extend the approach proposed by Lee et al. [8, 9] by adding QoS matching to webservice composition search. Their proposal named FSIDB (Full Solution Indexing using Database) uses a relational database approach for automatic

service composition. All possible service compositions are precomputed as paths in the graph along with their effective QoS values and stored in a relational database. When a user queries for a service composition, the system uses a SQL query to search for a service composition satisfying the user requirements.

Utkarsh [12] propose the development of a Web Service Management System (WSMS): a general-purpose system that enables clients to query multiple Webservices simultaneously in a transparent and integrated fashion. They build virtual tables for input/output parameters of webservices to manage service interfaces, and uses a multi-thread pipeline executive mechanism to improve the efficiency of webservices search, so the service composition problem is transformed into query optimization in the database. Query processing over webservices is visualized as a work flow or pipeline: input data is fed to the WSMS, and the WSMS processes this data through a sequence of webservices. The output of one Webservice is returned to the WSMS and then serves as Input to the next webservice in the pipeline, finally producing the query results.

5 Object-Relational Databases for Service Composition

In the previous section, we gave an overview of approaches that use a relational database to store webservice information. There are several advantages of utilizing a normalized database as

- Provides indexing.
- Minimizes modification anomalies.
- Saves space.
- Enforces referential integrity.
- Provides High Performance.

One reason to carefully review the usage of normalization in a database design is the database's intended use. There are certain scenarios where the benefits of database normalization are outweighed by its costs.

Two of these scenarios are described below

1. Immutable Data and Append-Only Scenarios—Database normalization may be unnecessary in situations where we are storing immutable data such as financial transactions or a particular day's price list.
2. When Multiple Joins are needed to produce a Commonly Accessed View—the biggest problem with normalization is that you end up with multiple tables representing what is conceptually a single item. With such design, it takes multiple SQL Join operations to access and display the information about a single item. This implies reduced database performance. To make a long story short, a normalized database in such a scenario requires much more CPU, memory, and I/O to process database queries. A normalized database must locate the requested tables and then join the data from the tables to either get the requested information or to process the desired data.

Table 4 Relational tables after further normalization

(a) WSInput table

WSIP	WS ID	InParsId
1	ws1	1
2	ws1	2
3	ws2	2
4	ws2	3
5	ws3	2
6	ws3	3
7	ws4	4
8	ws4	5
9	ws4	6
10	ws5	2
11	ws5	3
12	ws5	6

(b) WSOutput table

WSOP	WS ID	OutParsId
1	ws1	4
2	ws1	9
3	ws2	5
4	ws2	10
5	ws3	6
6	ws3	8
7	ws4	7
8	ws5	8

Service database schema fits the second scenario. Consider the type of queries that we plan to use for webservice composition

- List all webservices that match a given set of input parameters.
- List all webservices that match a given set of output parameters.
- List all webservices that are composable with a given set of output parameters.

Clearly, the tables in Table 4 would require multiple joins to support these types of queries. Also, the number of tuples in these tables increases drastically when the services have more input/output parameters, thereby generating more tuples in table joins, hence decreasing the database performance. Though the techniques discussed in the previous subsection use database technology for service matching and compositions, still those are constrained to the usage of multiple joins. Thus, we require a strategy to store these multivalued input and output parameters in the registry.

Object-relational database (ORDB) technology has emerged as a way of enhancing object-oriented features in relational database systems. In a relational model, multi-valued attributes are not allowed in the first normalization form. The solution to the problem is that each multiple-valued attribute is handled by forming a new

table or distributed across multiple rows of the same table. To retrieve the data back in such a storage design, one has to do multiple joins across the tables (or the table, if stored as multiple rows in the same table). To avoid the need for multiple joins and to speed up the query, we propose to use multi-valued attributes for storing input and output parameters in our database design, and hence use an Object-Relational Database for our proposed Extended Service Registry. ORDBMs allow multi-valued attributes to be created in a database by using *Collections Type* or *Nested Tables*. The advantage of this approach is that it supports querying for services efficiently and also supports complex queries involving input and output parameters.

For a given user query, the algorithms discussed so far generate a service composition based on **exact matches of input/output parameters**. Generation of such a service chain fails when the preceding service's output (ws_P^O) is not an exact match of input parameters of a succeeding service (ws_S^I). Our proposal alleviates this problem by making match criteria flexible, by introducing two more matches called the partial match and super match for conditions $ws_i^O \subset ws_j^I$ and $ws_i^O \supset ws_j^I$, respectively [13]. To avoid the overheads involved in multiple joins and to further speed up querying, our proposal uses multivalued attributes for the storage of input and output parameters. To support this structure our proposed Extended Service Registry [14] uses an object-relational database.

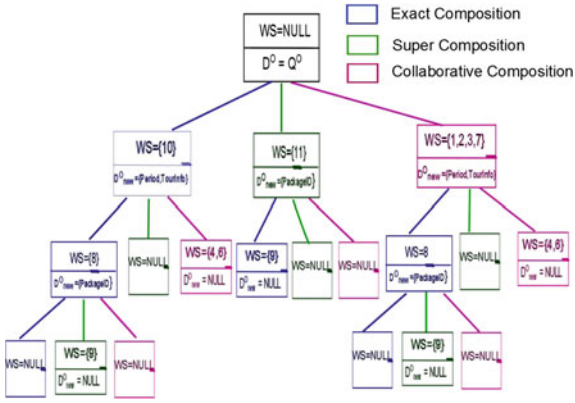
5.1 Service Matches and Service Composition Types

A webservice, ws, has typically two sets of parameters—set of input parameters ws^I and set of output parameters ws^O. The classical definition of service matching, i.e., one-to-one and onto mapping between input and output parameters of matching services, is extended to give rise to three types of service matches:

1. **Exact Match**: A webservice ws_i is an exact match of webservice ws_j if the input/output parameters of ws_i exactly match all the input/output parameters of ws_j.
2. **Partial Match**: A webservice ws_i is a partial match of webservice ws_j if the input/output parameters of ws_i partially match the input/output parameters of ws_j.
3. **Super Match**: A webservice ws_i is a super match of webservice ws_j if the input/output parameters of ws_i is a superset of the input/output parameters of ws_j.

Utilizing the three types of service matches we define three types of composition as

Fig. 5 Example CST



1. **Exact composition (EC)**: Exact composition is a composition obtained by using a webservice that is exactly matching with user required outputs.
2. **Super composition (SC)**: Super composition is a composition obtained by using a webservice that has a super match with user required outputs.
3. **Collaborative composition (CC)**: Collaborative composition is a composition obtained by using a set of partial matching webservices that can collaboratively satisfy the desired set of output parameters.

All the above composition types may require additional input parameters than those provided by the user.

5.2 Service Composition

In order to visualize the composition process and to find possible compositions that satisfy a given user query, we propose to construct a *composition search tree.*

Figure 5 depicts *composition search tree* generated when the webservices in Table 5 are used to construct the tree for a query with $Q^I = \{\text{Date; City}\}$ and $Q^O = \{\text{HotelName; FlightInfo; CarType; TourCost}\}$.

The process of constructing the composition search tree is given in Algorithm 2. The various notations used in the algorithm is described in Table 6.

Table 5 Example webservices

Service no.	Service name	Input parameters	Output parameters
ws1	HotelBooking	Period, City	HotelName, HotelCost
ws2	AirlineReservation	Date, City	FlightInfo, FlightCost
ws3	TaxiInfo	Date, City	CarType, TaxiCost
ws4	DisplayTourInfo	HotelName, FlightInfo, CarType	TourInfo
ws5	TaxiReservation	CarType, Date, City	TaxiCost
ws6	TourPeriod	Date, City	Period
ws7	TourCost	TourInfo	TourCost
ws8	AgentPackage	PackageID	Period, TourInfo
ws9	TourPackages	Date, City	PackageID
ws10	TourReservation	Period, TourInfo	HotelName, FlightInfo, CarType, TourCost
ws11	PackageDetails	PackageID	HotelName, HotelCost, FlightInfo, FlightCost, CarType, TaxiCost, TourCost

Table 6 Notations used in CST Algorithm

Notation	Description
Q^O	Set of output parameters desired in the searched webservice as given in the user query
Q^I	Set of input parameters that the user is capable of providing as given in the user query
WS	Webservice/Set of webservices satisfying the D^O of the Parent node
NWS	Number of webservices participating in the service composition
D^O	Desired set of output parameters
R^I_{EC}	Additional input parameters required to execute the Exact matching webservice
R^I_{SC}	Additional input parameters required to execute the Super matching webservice
R^I_{CC}	Additional input parameters required to execute the webservices participating in the Collaborative composition

Algorithm 2: Composition Search Tree Construction

```
   Input: WSInOutTable, Covering clusters,
   Q^O, Q^I Output: composition search tree
 1 Create a RootNode with D^O = Q^O
 2 NWS = 0 , WS = ø ;
 3 Insert the RootNode to LivenodesQ
 4 CurrentNode = LivenodesQ.Delete()
 5 Retrieve matching services from service clusters for CurrentNode //
    search for Matching services
 6 Classify the compositions
 7 repeat
 8     if ExactComposition then
           // Left child for EC
 9         Create LeftChild for CurrentNode
10         WS = ws
11         NWS = NWS + 1 , D^O = RI_EC^I
12         if D^O ≠ø ; then
13             LivenodesQ.Insert(LeftChild)
14         else
15             Mark the LeftChild as SolutionNode
16             Insert a copy of LeftChild to Solutions
17         Make the LeftChild point to its ParentNode
18     if SuperComposition then
           // Middle child for SC
19         Create MiddleChild for CurrentNode
20         WS = ws
21         NWS = NWS + 1 , D^O = RI_SC^I
22         if D^O ≠ø then
23             LivenodesQ.Insert(MiddleChild)
24         else
25             Mark the MiddleChild as SolutionNode
26             Insert a copy of MiddleChild to Solutions
27         Make the MiddleChild point to its ParentNode
28     if CollaborativeComposition then
           // Right child for CC
29         Create RightChild for CurrentNode
30         WS = WS_CC
31       NWS = NWS + | WS_CC | , D^O = RI_CC^I
32         if D^O ≠ø then
33             LivenodesQ.Insert(RightChild)
34         else
```

```
35              Mark the RightChild as SolutionNode
36              Insert a copy of RightChild to Solutions
37          Make the RightChild point to its ParentNode
38      if No Compositions then
39          Mark CurrentNode as UnsolvableNode
40      CurrentNode = LivenodesQ.Delete()
41      Extract compositions from Covering clusters // Solve for Additional
          input parameters
42  until LivenodesQ is not empty
```

The composition search tree finds all compositions satisfying the given query as explained in the previous section. The array Solutions stores all leaf nodes that give compositions satisfying the user query in the composition search tree. Compositions satisfying the given query can be obtained by tracing back from a leaf node till the root node in the composition search tree. The set of webservices required for service composition satisfying a given query for a solution node can be obtained using procedure *WebServicesInComposition*. Algorithm 3 lists all possible compositions for a given user query obtained from composition search tree generated for the query. Each solution is a set of webservices that need to be composed in an order, to obtain the desired output parameters utilizing input parameters provided by the user.

Algorithm 3: All Solutions in CST

```
Input: Solutions
Output: SWS for all solutions in CST
 // SWS is a set of required services for composition
1 SWS = ø ;.
2 for each Leaf Node in Solutions do
      // Call procedure WebServicesInComposition for each Leaf Node
3     SWS=WebServicesInComposition(Leaf Node)
4     Print SWS
```

From the CST depicted in Fig. 5, we have five solutions for the given query, listed below in Table 7.

Table 7 Solutions in CST

Sl no.	Webservices in composition	Composition types involved
1	ws9, ws8, ws10	Exact, super
2	ws4, ws6, ws10	Exact, collaborative
3	ws9, ws11	Super
4	ws9, ws8, ws1, ws2, ws3, ws7	Collaborative, super
5	ws4, ws6, ws1, ws2, ws3, ws7	Collaborative

```
Procedure WebServicesInComposition
   Input: Leaf Node
   Output: SWS for service composition
    // SWS is a set of required services for composition
1 SWS = ;.
2 CurrentNode =LeafNode with a solution for composition
3  while CurrentNode 6= RootNode do
        //   WS is the set of services in the CurrentNode.
4 SWS = SWS [ WS
5 CurrentNode = Parent Node of the CurrentNode
6 return [SWS]
```

6 Conclusion

In this chapter, we introduce the concepts of service composition and give a brief description of the various classifications of service composition. We then explain the various approaches proposed so far for service composition, with an emphasis on input/output parameter based composition. A detailed description of the latest approaches for service composition, relational and object-relational databases, the research proposals in these areas are further discussed. It is evident from the discussion that database based approaches are more promising in that they are scalable and efficient when compared to graph-based approaches.

References

1. Yang Syu, Shang-Pin Ma, Jong-Yin Kuo, and Yong-Yi FanJiang. A survey on auto-mated service composition methods and related techniques. In Services Computing (SCC), 2012 IEEE Ninth International Conference on, pages 290–297. IEEE, 2012.
2. Pablo Rodriguez-Mier, Manuel Mucientes, and Manuel Lama. Automatic web ser-vice composition with a heuristic-based search algorithm. In Web Services (ICWS), 2011 IEEE International Conference on, pages 81–88. IEEE, 2011.
3. I. Budak Arpinar, Boanerges Aleman-Meza, Ruoyan Zhang, and Angela Maduko. Ontology-driven web services composition platform. In Proceedings of the IEEE International Conference on E-Commerce Technology, CEC '04, pages 146–152, 2004.
4. John Gekas and Maria Fasli. Automatic web service composition based on graph network analysis metrics. In On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, volume 3761 of Lecture Notes in Computer Science, pages 1571–1587. Springer, 2005.
5. S.V. Hashemian and F. Mavaddat. A graph-based framework for composition of stateless web services. In Web Services, 2006. ECOWS '06. 4th European Conference on, pages 75–86, 2006.
6. Hassina Nacer Talantikite, Djamil Aissani, and Nacer Boudjlida. Semantic annotations for web services discovery and composition. Computer Standards Interfaces, 31:1108–1117, 2009.
7. M.B. Blake, W. Cheung, M.C. Jaeger, and A. Wombacher. Wsc-06: The web service challenge. In E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference on, pages 62–62, 2006.
8. Joonho Kwon, Kyuho Park, Daewook Lee, and Sukho Lee. Psr: Pre-computing solutions in rdbms for fastweb services composition search. In Web Services, 2007. ICWS 2007. IEEE International Conference on, pages 808–815, July 2007.
9. Daewook Lee, Joonho Kwon, Sangjun Lee, Seog Park, and Bonghee Hong. Scalable and efficient web services composition based on a relational database. J. Syst. Softw., 84(12), 2011.
10. Cheng Zeng, Weijie Ou, Yi Zheng, and Dong Han. Efficient web service composition and intelligent search based on relational database. In Information Science and Applications (ICISA), 2010 International Conference on, pages 1–8, April 2010.
11. Jing Li, Yuhong Yan, and Daniel Lemire. Full solution indexing using database for qos-aware web service composition. In Proceedings of the 2014 IEEE International Conference on Services Computing, SCC '14, pages 99–106, 2014.
12. Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. In Proceedings of the 32Nd International Conference on Very Large Data Bases, VLDB '06, pages 355–366, 2006.
13. H.N. Lakshmi and H. Mohanty. Rdbms for service repository and composition. In Advanced Computing (ICoAC), 2012 Fourth International Conference on, pages 1–8, Dec 2012.
14. Lakshmi H.N. and Mohanty H. Extended service registry to support i/o parameter-based service search. In 1st International Conference on Intelligent Computing, Communication and Devices, ICCD 2014, pages 145–155, 2015.

Handling Faults in Composite Webservices

Vani Vathsala Atluri and Hrushikesha Mohanty

Abstract Webservices have been instrumental in implementing business processes. Individual services available over the web can be composed into a bigger service to realise complex business processes. In order to provide resilient execution for composite services, fault handling must be provided. In literature, various strategies of handling faults in webservices can be found. In this chapter, we briefly describe different types of strategies applicable to webservices for handling faults with focus on transient faults.

Keywords Webservices · Choreography · Checkpointing · Fault handling

1 Introduction to Webservices

A software module that offers services over the internet can be called as a webservice. Webservices are witnessing enormous growth nowadays. This growth is because of availability of a large number of services on the internet and also for their operational use and efficiency. The ease of operation is because of Service-Oriented Architecture (SOA) that provides a framework to host as well as to utilise an available service.

Service-Oriented Architecture is a form of distributed systems architecture [1] which is basically characterised by separation of service specification from service implementation. The main stakeholders in this architecture are provider of service, consumer of service and a services registry. Service providers who want to make their services public publish about their service in a registry. Service consumers search for required services in the registry and invoke a suitable service. Figure 1 depicts this service-oriented architecture.

V. V. Atluri (✉) · H. Mohanty
CVR College of Engineering and University of Hyderabad, Hyderabad, India
e-mail: atlurivv@yahoo.com

H. Mohanty
e-mail: mohanty.hcu@gmail.com

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_5

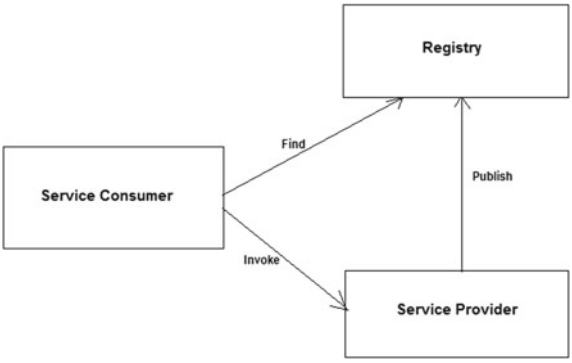


Fig. 1 Service-oriented architecture

Webservices have the following properties: (1) Only a logical view of actual programmes (called operations) which implement business processes is made public. The logical view defines what these operations do but does not specify how they do. (2) A published service operation can be invoked by a service consumer using platform independent messages (called interactions). These messages and operations are described in a document called as Web Services Description Language (WSDL) which is published in the registry. These properties enable webservices to provide reusable business functionality irrespective of the platform on which they are working.

A webservice that implements a business process, without invoking another webservice, is called as atomic webservice. A simple business process may be realised by invoking only one webservice. In order to accomplish complex business processes, suitable atomic webservices are searched over the internet and are composed into a bigger application called composite application. These composite applications may expose themselves as services on the web, resulting in what are known as service compositions or composite services. The webservices which participate in such a composition are called as constituent services or, participants.

Two main jargons used frequently in the context of composite webservices are Choreography and Orchestration. In the following subsection, we introduce and differentiate them for ease of understanding of concepts that follow.

1.1 Types of Composite Services

Choreography and Orchestration are the two words which are used most often with reference to webservice composition. The word Choreography is used to describe the collaboration between multiple services which communicate with one another so as to achieve a common business process. This description includes a list of roles of participating services, sequence in which messages are to be exchanged for

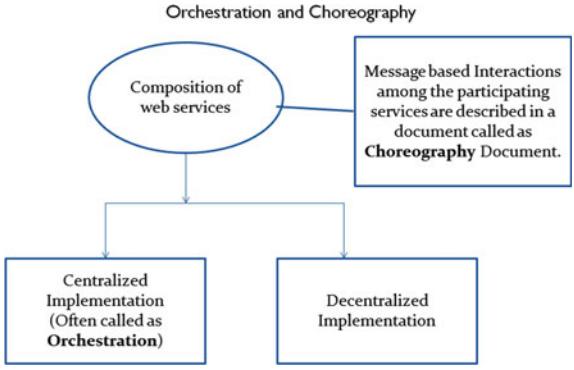


Fig. 2 Orchestration and choreography of webservices

interaction among each other and data elements to be exchanged. But choreography does not describe the implementation details of any individual service.

The described choreography model may be implemented in two ways (refer to Fig. 2): (1) There is a central coordinator which monitors and dictates the order in which constituent services are to be called. Such a composition with a central coordinator is called as an Orchestration. Centralised orchestrations are to be used when amount of data to be transferred among constituent services is not large. But if large amount of data is to be transmitted, or if such a centralised arrangement is not implementable due to limitations imposed by business rules, implementations of complex business processes have to be decentralised. (2) In decentralised implementation, all the participating services are treated equally, with no central coordinator controlling the sequence of interactions. All the constituent services have a local view of interactions that they perform with other constituent services. No single service has a global view of the overall integration.

For providing seamless operation of composite webservices over unreliable Internet, webservices have to be outfitted with efficient fault handling strategies. In the following section, we introduce and discuss this issue.

2 Faults and Their Handling in Webservices

Due to the supply of several functionally equivalent services over the internet, it becomes crucial for webservices to use a robust fault handling strategy to sustain in the competition. The fault handling strategies become indispensable as webservices execute over the internet which is highly dynamic.

A webservice is said to fail when it does not deliver expected service or what it delivers does not match with what is requested. A fault is defined as a defect in the environment of a webservice or in its code which results in a failure of the service [2]. Webservice faults can be categorised as follows [3, 4]:

- Content/Development faults: Bugs that might have been introduced in design or in code result in development faults. As a result of these faults, the service gives incorrect results. A price query service gives wrong value back for a given Item ID.
- Domain-specific faults: Faults of this type are defined by application designers of a service and are published in WSDL document of that service. These faults are raised when the service halts unsuccessfully. A student cannot be granted admission due to his ineligible qualification.
- Transient/Temporary hardware faults: These faults occur due to temporary server crashes or due to the failure of power or connection link. These faults are asynchronous, and self-healing techniques like rollback and recovery enable the system to carry forward work.
- Permanent hardware faults: Irreparable faults such as crashing down of a server hosting a service result in this kind of faults.
- Interaction faults: The faults due to differences in actual parameters and expected parameters in messages exchanged between service provider and service consumers come under interaction faults.
- SLA faults: Service Level Agreement (SLA) specifies deadlines for expected response time, cost of service, reliability etc., upon which both the service consumer and service provider agree. These faults occur when an invoked service does not satisfy SLA. For a particular service query, the response is given in 20 s but the value specified in Service Level Agreement is just 5 s.

Content faults and interaction faults need modification in code without which they are bound to resurface. Domain-specific faults are various situations that arise in business applications. Composite webservice development languages like BPEL provide fault (exception) and event handlers which provide alternate execution paths for every predefined fault/event that occurs during execution. The domain-specific faults and corresponding actions to be taken to handle those faults have to be specified explicitly by application designers.

Permanent hardware faults are inadvertent and unavoidable in nature. Hence we have to look out for ways and means of handling permanent hardware faults. Same is the case with transient faults. When either of these two faults terminates the execution of a webservice, the service is to be quickly restored back for its continuation. But this recovery has to be done in a manner that is transparent to the user. These faults may result in SLA faults when recovery actions take considerable additional time and cost. Webservices have to take necessary steps to avoid SLA faults (violations). Thus, it becomes very important for service providers to avoid or reduce possible delays generated due to recovery from transient faults or permanent faults. Thus, fault handling techniques are of importance.

In this chapter, we present our survey on fault handling techniques proposed in the field of webservices for handling transient faults and permanent faults. In the following section, we introduce two different fault handling strategies that have been applied to webservices.

3 Transient Fault Handling Strategies for Webservices

Transient fault handling techniques proposed in the literature fall under two basic categories [2]: backward recovery and forward recovery. Upon failure of a service during its execution, the service is rolled back to an earlier saved state, also called as checkpoint, and then the strategy is called as backward recovery. Execution of the services resumes from this saved state. On the contrary, if the failed service resumes its execution by trying out alternative paths instead of going back to a check pointed state, then the strategy is called as forward recovery. The popular examples for forward recovery techniques include substitution (invoking functionally equivalent services) [4–7] and fault compensation.

In the following subsections, we discuss each of them and their applicability.

3.1 Backward Recovery

Checkpointing has been used in previous decades very successfully for recovering database applications after their failure. Checkpointing is a technique that has proved itself time and again as a robust fault handling strategy across areas like distributed computing, operating systems, banking transactions, etc. Checkpointing is a proactive technique which prescribes to save the state of an application so as to enable its recovery in case of any failure of the application at a later time. A failed application rolls back to a previously checkpointed state and continues its execution from there on. Checkpointing and recovery can be applied to webservices also due to the following reasons:

- Checkpointing and recovery scheme is independent of application, environment or platform and hence can be applied to webservices too.
- As transient faults do not surface at the same location in every execution, backward recovery technique of rolling back the failed service to a previous state and resuming its execution from there on proves to be an efficient strategy for recovery from transient faults.
- For another type of webservice fault, *permanent server crash*, the process of checkpointing a service and in the case of failure of the primary server, migrating the service instances onto a secondary server/redundant server and then resuming the service instances will largely reduce the amount of rework.

3.2 Forward Recovery

Popular forward recovery techniques that are mostly used in webservices are substitution and fault compensation. When a service is invoked by another webservice (say caller), it is possible that either the caller or the invoked service (also called as callee) fail. In the event of failure of the callee, backward recovery using checkpointing may

be employed by the caller. Or, alternatively, if the callee does not reply back within a stipulated time, the caller might resort to calling an alternate functionally similar service instead of waiting for a reply from the initially invoked callee. This approach is called as Substitution.

Fault compensation specifies corrective actions that negate the effect of an already executed unsuccessful activity. In cases where re-execution helps completing an activity, compensation is not the required solution.

But, in case the server hosting the caller itself fails there might be many instances of the service which are still in execution at the time of server crash. (A service instance is a webservice in execution. When two or more requests for a webservice are received simultaneously from users, multiple instances would be initiated). In the absence of checkpointing and recovery, all the caller instances have to restart their execution. This re-execution will trigger re-invocation of the callee by all restarted caller instances, thereby increasing the net execution times. Hence if the caller fails, substitution approach is not practically suitable. Instead, checkpointing the caller at places of invocation of partner services and rolling back to one of these checkpoints without re-invoking partner services greatly improves the performance of failed executions.

4 Checkpointing and Recovery

In this section, we introduce the basic terminology generally used while discussing checkpointing techniques and, discuss various techniques proposed in literature for checkpointing distributed systems and their applicability to webservices.

While checkpointing standalone applications, the focus would be on maintaining a balance between (i) recovery overhead that will be incurred during failure and recovery and (ii) checkpointing overhead during normal executions of the application. Several strategies have been worked out in the past in the area of checkpointing individual applications. Important among them are (1) *Checkpoint at regular intervals of time* (2) *Checkpoint after every 'n' lines of code* and (3) *checkpoint any crucial work done*.

Multiple process running concurrently at geographically distant locations communicate with each other mainly through messages in what are called as *distributed applications*. To checkpoint processes in such applications, the above-mentioned strategies are insufficient. Additionally, event-driven checkpointing is triggered by two main events: sending and receipt of a message is employed to checkpoint-distributed applications.

Checkpointing strategies [8–17] on distributed applications revolve around one main concept: consistent checkpoints. A set of local checkpoints of processes constituting the distributed application is called as a global checkpoint. A global checkpoint is called as a consistent checkpoint if it does not lead to loss of a message or any information and there are no redundant messages. In other words, a global checkpoint is said to be consistent if it does not result in any orphan messages. A message

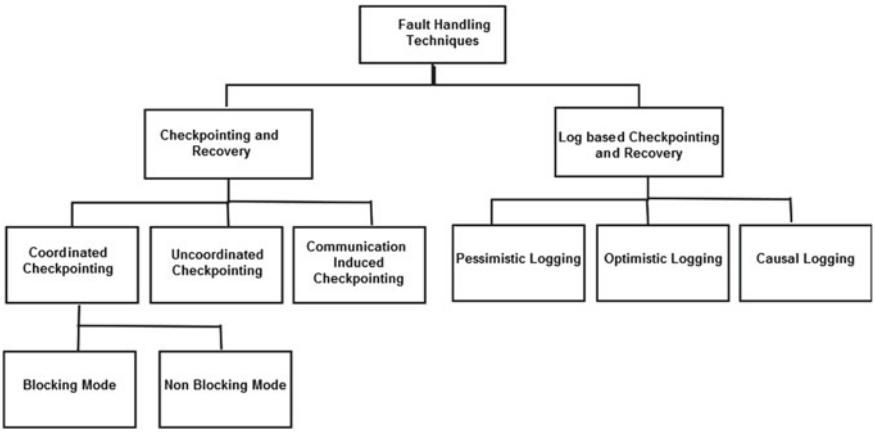


Fig. 3 Classification of fault handling techniques in distributed systems

is said to be an orphan message if it is recorded as delivered but not recorded as sent by their respective processes after recovery to a global checkpoint.

On the other hand, a global checkpoint is not inconsistent if a message is recorded as sent but not recorded as delivered by their respective processes, because this situation is identical to a state, wherein the message is sent by its sender and is in transit. All the recovery protocols require processes to save the incoming messages on stable storage for possible replay during recovery.

In the following subsection, we discuss various distributed checkpointing techniques proposed in literature and their applicability to webservices.

4.1 Applicability of Distributed Checkpointing Techniques to Webservices

Fault handling techniques in distributed computing can be categorised into pure checkpointing and recovery techniques or log based checkpointing and recovery (refer to Fig. 3). As discussed in [10], the first category of techniques may be further subdivided into: *coordinated checkpointing, uncoordinated checkpointing* and *communication-induced* checkpointing. As per the first two techniques, apart from the failed process, other processes involved in communication with the failed process also need to be rolled back. In yet another backward recovery strategy, checkpointing is coupled with message logging.

A central monitor is the heart of coordinated checkpointing, which would be entrusted with the job of coordinating and issuing control messages to all the individual processes of the distributed application. Using these control messages, the central monitor would trigger checkpointing of all the processes. There are two main variants of coordinated checkpointing: blocking mode coordinated checkpointing [18] and nonblocking mode coordinated checkpointing [8–10, 17]. A brief outline of blocking mode coordinated checkpointing algorithms is given in Algorithm 1.

Algorithm 1: Coordinated Checkpointing : Blocking Mode

1: **Participants:** All processes that are part of application, one among them designated as Central initiator.
2: Central initiator takes a tentative checkpoint.
3: Central initiator sends control messages to all other processes to take checkpoints.
4: Upon receiving control message each process does the following things in the given sequence:
    i) Stop computation
    ii) Stop sending and receiving messages
    iii) Take a tentative checkpoint
    iv) Send back an acknowledgement message to the initiator
5: Upon receiving acknowledgement messages from all processes, initiator sends commit messages to them.
6: Upon receiving commit message from initiator, each process converts its tentative checkpoint to permanent checkpoint and comes out of blocking mode to continue its execution.

The key strategy employed by all blocking algorithms is to see that participating processes do not send or receive messages once checkpointing phase begins. This restriction is imposed in order to make sure that the global checkpoint taken is consistent. But this strategy has got a major disadvantage of suspending/blocking all the executing processes for a considerable amount of time which proves to be costly in terms of execution speeds.

To prevent temporary blocking of processes and to enable them to receive and send messages, Chandy and Lamport [9] have proposed the nonblocking approach of taking coordinated checkpoints, which is outlined in Algorithm 2.

Algorithm 2: Coordinated Checkpointing : Non Blocking Mode

1: **Participants:** All processes that are part of application, one among them designated as Central initiator.
2: Central initiator takes a checkpoint.
3: Central initiator sends marker messages (checkpoint request messages encoded with serial no of global checkpoint to be taken) to all other processes.
4: Upon receiving a marker message, each process gets to know about the initiation of a checkpoint phase.
5: In this phase, before sending any application message, each process does the following things in the given sequence:
    i)Take a checkpoint
    ii)Transmit a marker message to whom an application message has to be sent
    iii)Send the application message

This approach ensures that any global checkpoint taken is consistent. If only the initiator transmits marker messages, the following scenario might occur: Process A receives a marker message from its initiator and takes a checkpoint. Then, it sends an application message *m* to process B. Process B receives the message *m* first and then, a marker message from its initiator. Process B takes its checkpoint. Hence in the checkpoint taken by process A, sending of message *m* is not recorded, whereas in the checkpoint taken by process B, receipt of message *m* is recorded, resulting in an orphan message and hence an inconsistent checkpoint. Instead if process A transmits a marker message to process B, before the message *m*, process B would first take a checkpoint and then receive the message *m*. This approach results in nonblocking consistent checkpointing. This advantage comes with a cost: processes have to circulate more number of control messages among them to take a consistent checkpoint.

As pointed out earlier, both blocking mode and nonblocking mode algorithms of coordinated checkpointing dictate that all the processes, irrespective of which process has failed, have to be rolled back to their latest checkpoints to maintain a globally consistent state.

On the contrary, processes in uncoordinated checkpointing [10, 19] take their checkpoints decisions on their own without requiring any central monitor. To take a consistent global checkpoint, each of them shares dependencies among each other (order of sending and receiving messages) using data structures like dependency graphs. But the disadvantage comes when the processes have to coordinate with each other to compute a global recovery line when one of the processes fails. This coordination is achieved by exchanging additional control messages. Set of local checkpoints that belong to a global recovery line are decided using dependency graphs. The processes whose local checkpoints belong to the computed global recovery line are forced to roll back to their latest checkpoints. Thus, processes lose their independence. More dangerously, these forced rollbacks might lead to a more adverse effect known as domino effect [10]. Domino effect is a cascading effect that occurs if processes are more interdependent on each other, and rollback of a set of processes will trigger a sequence of rollbacks in which all the processes have to start their execution from the beginning.

Yet another type of checkpointing algorithms have been proposed in literature called Communication-induced checkpointing [15, 20, 21] algorithms. As per these algorithms, checkpointing details are piggybacked by each process in the messages sent to other processes of the application. The decision of whether to take a checkpoint or not is taken by the receiver of these messages after looking at the piggybacked checkpointing information. In log-based recovery, we come to see a new type of checkpoints called useless checkpoints (checkpoints that do not contribute to a global consistent state). The key approach used by these algorithms is not to allow some predefined patterns in communication (like Z-Paths and Z-Cycles [15]). Although these algorithms avoid domino effect, they also require some of the other processes to rollback in the event of failure of one of them, to maintain consistent state.

All the three types of checkpointing techniques discussed above require several processes to rollback in the case of failure and recovery. In composite webservices,

forcing the rollback of other constituent services during execution is a practical proposition. Particularly for service-oriented applications, rollback of several services affects resiliency in service delivery.

Log-based recovery [10, 22–25] is similar to uncoordinated checkpointing when it comes to independence of processes in taking checkpoints. Additionally, processes are required to log every message received by them in the case of log-based recovery. These logged messages are replayed for the recovery of the process when it fails and rollbacks to its last saved checkpoint. Other processes are not forced to rollback. On the contrary, they can continue their execution as far as possible (till the point where they are waiting for a message from the failed process). Thus, rollback has to be done only by the process that has failed in message logging technique.

Log-based recovery can be broadly categorised into three flavours [10]: *Pessimistic logging, Optimistic logging and Causal logging*. According to Pessimistic logging [24] when a message is received, execution of the process has to be suspended temporarily, the message has to be logged on to a permanent storage, and then the message can be processed by the process. Blocking of the process, although temporarily, leads to performance overhead. In order to avoid the performance overhead incurred in pessimistic logging, the approach of asynchronous logging of messages is adopted by optimistic logging [23, 24]. When messages are being logged asynchronously, the process can be executed parallely with logging. As a result of asynchronous logging, optimistic logging leads to the creation of orphan processes [10] (a process whose state depends on a message which has not been saved). Contrary to pessimistic logging, all the processes which have received messages from orphan process before its failure are also rolled back. Thus, optimistic logging requires additional processes to be rolled back in addition to failed process. The main feature of optimistic logging is that it ensures total removal of orphans by the end of recovery process. Processes that are not involved in recovery resend the messages delivered to and not saved at the failed process, thereby aiding the failed process in its recovery.

Causal logging [10, 22] takes on the advantages of both pessimistic and optimistic logging. Asynchronous logging of messages is adopted from optimistic logging. The underlying principle of pessimistic logging is that it requires the messages sent to a process be logged before proceeding to send a message to remaining processes, thereby avoiding orphan messages. The same principle is adopted by causal logging. These advantages come with a disadvantage, i.e. recovery of the failed process involves complex procedure. Surviving processes must resend the unlogged messages, along with their order of replay, to aid in recovery of the failed process. In order to resend messages in the hour of need, each process must know the global order of message replay. Due to this reason, each process maintains a dependency graph that depicts the causal ordering of messages. These dependency graphs are exchanged among processes that aid other processes involved in a recovery. *This exchange of graphs is inevitably a substantial overhead, which is a hindrance to providing quality service and hence cannot be applied to a webservice if it has to honour an SLA.*

To handle failures within coordinated workflows that share common resources over a network, the authors of [13] propose a new scheme for compensation and re-

execution which eliminates unnecessary recovery overheads when different modules (called steps) of a workflow are rolled back partially and re-executed. The focus of this work is to maintain consistency of data items that are shared among workflows. Decision on whether to re-execute or to compensate a step depends upon the updation of involved data items values in that step. This work does not address the issue of meeting time and cost deadlines which are central to webservices checkpointing.

When multiple webservices work in tandem with one another to form a composition, failure in one of the webservices should not require rollback of other partner webservices during recovery. Checkpointing coupled with log-based recovery may seem to be applicable to webservices as it does not require surviving processes to rollback. But the associated costs (unavoidable logging of messages, creation of orphan processes and exchange of dependency graphs) of the three log-based recovery approaches would reduce their applicability to webservices.

Hence, an approach for checkpointing and recovery of webservices should implement the following: (1) not to request rollback of processes other than failed process (2) use pessimistic logging to avoid creation of orphan processes but not to log all the messages (since it results in considerable performance overhead) and (3) to consider the characteristics of webservices in designing a strategy.

4.2 Checkpointing Webservices

Having discussed the non-applicability of distributed checkpointing techniques to webservices in the previous section, we now present important characteristics of webservices which every webservice checkpointing scheme has to consider while deciding on checkpoint locations:

1. **Composite nature of webservices and non-repeatability of actions**: Constituent services of a composite service interact with each other in a specified sequence. Checkpoint locations should be such that rollback to a checkpoint should not result in re-invocation of other webservices that perform nonrepeatable actions for business obligations. For example, in Fig. 4, online booking service WS1 sends payment details and invokes another webservice, theatre booking webservice WS2 which performs a nonrepeatable action of booking a seat in a specified theatre. A failure of WS1 at time T1 would require a rollback of WS1 to one of its checkpoint locations. Rollback to checkpoint at location L1 results in re-invocation of WS2 (not desirable), whereas rollback to checkpoint at location L2 does not result in re-invocation of WS2 (desirable). Hence, checkpoint locations should be based on interactions between webservices.
2. **Compliance to SLA**: Webservices must work in compliance with Service Level Agreements (SLA). In order to see that a composite webservice does not violate its SLA, it is very important to plan its checkpointing and recovery based upon the promised QoS values of its constituent services.

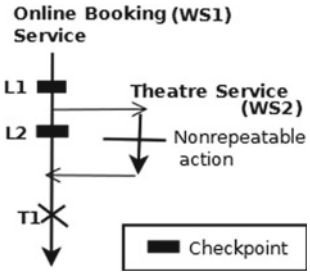


Fig. 4 Checkpoint locations and their impact

3. **Dynamic selection of constituent services**: One of the salient features of composite webservices is that some of the participating services can be selected at runtime resulting in dynamic composition of webservices. Considering their QoS values while placing checkpoints is vital in avoiding SLA faults that might arise during failed and recovered executions. Hence, checkpointing locations have to be revised after consider the QoS values of dynamically selected services.
4. **Dynamic nature of Internet and web server environments**: The traffic that flows through the internet is highly dynamic in nature which might result in large variations in the response times of webservices. Also, number of pending requests at a web server will have an impact on how quickly a request is processed by the web server. This dynamic nature of the environment in which webservices operate also has to be considered while taking checkpointing decisions.
   In literature, we can find some interesting papers [7, 26–30] that discuss the need and techniques of checkpointing webservices for their resilient execution. In this section, we discuss significant checkpointing works proposed for handling faults in webservices, analysing their merits and demerits. At the end of this section we present a comparison of these works based on the above-listed characteristics that each of them have considered while proposing a solution. Another criterion is also used for comparing these works.
5. **Decision on checkpointing locations**: Location of checkpoints in a composite webservice has impounding effect on total execution time for failure free executions and on recovery time for failed and recovered executions. Hence, proper placement of checkpoints is vital for resilient execution of webservices.

In the comparison table presented at the end of this section, we have specified whether each of the works has proposed any checkpointing algorithm or they left it to designers of applications to decide.

In one of the most cited webservice checkpointing works by Soumaya Marzouk et al. [27], the authors propose a fault handling mechanism to handle permanent server failures. The main contributions of this work are: (i) An architecture and (ii) A scheme for webservice recovery from server faults. The proposed architecture mainly consists of two servers, namely one primary and the other redundant secondary server to manage the permanent failure of primary server. If permanent server failure is not handled, it might lead to another serious problem of QoS vio-

lation. To avoid this situation, they propose to checkpoint the executing services instances and migrate executing instances from primary server to secondary server in case of failure of the former. Additionally, they introduce the use of two services that manage checkpointing and invocation of webservices. The first service is Web Service Checkpoint Manager (WSCM) and the second service is Web Services Invocation Manager (WSIM).

WSCM is responsible for checkpointing of webservices and also for their subsequent migration and recovery following the primary server crash. Suspended instances would resume their execution from their last saved checkpoints. After migration, the job of routing requests from service consumers to secondary web server is taken up by WSIM. Hence, WSCM and WSIM together achieve mobility of services. The approach of modelling the mobility, checkpointing and recovery and migrating as aspects is what makes their work stand apart.

Although several options for deciding checkpoint locations (like checkpointing at regular intervals of time, checkpointing just before communicating with other services, etc.) have been suggested by the authors, they leave it to the users to select a suitable option based on their requirement. Thus, the expertise of the users in placing checkpoints at apt locations decides the checkpointing and recovery overhead. Execution of the two services WSCM and WSIM on reliable servers is another performance overhead.

A recovery scheme for composite webservices has been proposed by Marta Rukoz et al. in [28]. Their main contributions include (i) An architecture and (ii) A recovery scheme for handling faults in a composite webservice. Their architecture consists of three layers: client, server and a software layer comprising of execution engine and engine threads. The states of execution of each of the participant services are monitored and captured by these software components. In case of failure of a service, all the surviving constituent services are allowed to proceed till the point where they need to communicate with the faulty service. They will be checkpointed at the state where they cannot proceed further. As soon as the failed service resumes, all the remaining, waiting services are also resumed. But this approach also requires the failed service to restart from the beginning since it is not checkpointed anywhere.

The authors of [29] propose a checkpointing and recovery scheme for Inter Organisational Workflows (IOWS). Their solution is to checkpoint an IOWS at every Maximum Sequential Path (MSP). A Maximum Sequential Path is defined as a set of all tasks in an IOWS that can be executed sequentially. A checkpoint is placed after every MSP. As every MSP represents a logical unit of work, the end of every MSP is argued as a suitable location to which the failed workflow can be reverted back. But the drawback of this proposal is that it results in large number of checkpoints whenever there are too many control flow branches.

Urban Susan et al. have proposed an interesting work in [30] to respond to execution errors in composite webservices. Their main contributions include a scheme for checkpointing and recovering orchestrated webservices. They introduce what are called as Assurance Points (APs) which act as both logical and physical checkpoints. At every AP marked at the end of a logical work in a choreography, a physical checkpoint is taken, but after checking with post and pre-conditions dictated by business

level constraints. Each AP is associated with a set of integration rules which specify whether the recovery is backward recovery or forward recovery. The decision is taken as per specified pre-conditions and post-conditions. Backward recovery resumes the execution from last AP, whereas, alternate paths are suggested in forward recovery. Even in their scheme, the responsibility of placing the assurance points is vested in the hands of the designer and there is no policy using which automatic placements of assurance points can be done.

In [31, 32], the authors propose a two-stage checkpointing: Design time and Deployment time checkpointing to handle transient faults in composite webservices. Design Time Checkpointing: In [31], the authors have captured interaction centric design of a choreographed webservice by proposing an interaction pattern model. Five atomic interaction patterns and five composition operators are proposed that are used to model essential elements of a choreography. Probable checkpointing locations are identified using the proposed patterns which are termed as C-points. Design time checkpointing rules are proposed that convert these C-points into checkpoints to avoid re-invocation of webservices that perform nonrepeatable actions (*Static Checkpointing*). They also advocate recovery rules that ensure recovery without initiating re-invocation of other services that perform nonrepeatable actions.

Deployment Time Checkpointing: In [31], the authors present their proposed second stage of checkpointing. Response time, vulnerability and cost of service are identified as the primary QoS attributes which have to be considered while taking checkpointing decisions. The proposed interaction pattern model in [31] is amended to include modelling of QoS attributes. Using these attributes and other quantities measurable at deployment time like: time taken to checkpoint, time taken to log a message and, restoration time, etc., the proposed time and cost aware checkpointing algorithm marks checkpoint locations in a composite service. This algorithm recommends to take a new checkpoint only when time and cost constraints are not met when the composite webservice fails and recovers.

4.3 Comparison

In this subsection, we present a summary of our survey. We present the strengths and weaknesses of all the surveyed papers on fault handling in composite webservices in the table below (Table 1). We compare them based on the features of composite webservices that they have considered while taking checkpointing decisions. We also consider whether checkpoint locations are automatically generated by the scheme or require user to specify them. It may be noted here that user-specified checkpoint locations require the user to have thorough knowledge of the application domain.

Table 1 Comparison of webservices checkpointing

	Non-repeatability of actions	Compliance to SLA (Time and Cost)	Dynamic selection of constituent services	Dynamic nature of the environment	Checkpoint locations (A/U)
Markouz et al. [27]	X	X	X	Y	U
Sen et al. [29]	X	X	X	X	A
Mansour et al. [7]	X	Only Time	X	Y	A
Susan et al. [30]	Y	X	X	X	U
Rukoz et al. [28]	Y	Y	X	X	U
Vani et al. [31, 32]	Y	Y	X	X	A

X Not addressed, *Y* Addressed, *A* Automatic, *U* User defined

5 Substitution

Substitution is another popular technique that is used for handling faults in composite webservices [4–7].

The approach proposed by Mansour and Dillon et al. in [7] consists of a central coordinator for handling faults in composite webservices. In order to aid in management of faults in a composite webservice, the authors propose to associate each webservice with a reliability factor. Reliability of every webservice is computed dynamically. A choreography of webservices that represents a composite webservice is pictorially represented in the form of a graph. Each participating service is represented by a vertex in the graph. If the execution of service a is followed by service b in the given choreography, then it is represented by placing an edge from vertex a to vertex b. For every edge (a, b) inserted between the services a and b, a measure $E(a, b)$ is defined as $E\ A\ e$ where E is the maximum expected time of recovery from the latest checkpoint, A is the recorded actual time of recovery and e is the expected total time of execution of service b.

A checkpoint is inserted after a and before b if the computed $E(a, b)$ is less than zero. If it is positive, then testing continues with the next edge to decide whether a checkpoint can be placed after that edge. In case of failure of a called webservice, the calling service is rolled back to its latest saved state and execution resumes from there. Then, an alternate functionally equivalent service is invoked. Apart from reliability, no other QoS attributes have a role in their model. Checkpoint locations are decided purely based on computed reliability values. The involvement of the network that lies between the geographically separated webservices and the environment in which services are deployed are totally ignored in deciding checkpoint locations.

In [4], yet again, reliability of webservices is used for handling faults of partner services in a composite webservice. Content faults, system faults, logical faults and service level agreement faults are the four types of faults that might appear in called services as identified by the authors. Replicate, notify, retry, ignore and wait are some of the eight different fault handling techniques which is proposed to encounter the faults specified above. Event Condition Action (ECA) rules are proposed that describe which fault handling strategy is to be used when and how. However, no strategy is proposed to handle faults in central coordinator that invokes partner services in a composite webservice.

The fault-tolerant strategy proposed in [6] also is an example of substitution. RobustBPEL is the framework proposed by the authors which is capable of automatically producing a fault-tolerant process equivalent to a given BPEL process. A failure in one of the constituent webservices results in the failure of the entire composite webservice. To avoid such a situation, a proxy server is kept available all through. In case of failure of primary server, all unanswered requests are routed to this proxy server. The job of forwarding requests is taken care by an adapt-ready process. The proxy server is responsible for calling a functionally equivalent service. RobustBPEL comes in two flavours. Static RobustBPEL always routes calls to the same set of functionally equivalent services which are hardwired in the code. Dynamic RobustBPEL, on the contrary, searches for equivalent services at the time of execution and routes calls to them.

In [5], the authors present a framework for self-healing of webservices. They have proposed an approach which is non-intrusive for recording message exchanges between the participants of a composite webservice. A four-step procedure is devised to monitor, diagnose and detect, plan repair and execute repair actions for providing self-healing. The SOAP messages are attached with QoS parameters and their values observed by monitor components running at receiver and provider ends. When a performance degradation is detected using a specified policy, alternate partner services are invoked. Dynamic binding and substitution are the main policies upon which their algorithm works.

In the next section, we discuss another fault handling strategy in webservices: Redundancy.

6 Redundancy

By and large, the technique used in this strategy is to maintain redundant web servers so that failure of one web server does not affect the delivery of a prompt service.

The work proposed in [33] addresses the problem of handling coordinator failures by providing an infrastructure called WS-Replication for WAN replication of webservices. Webservices are maintained on redundant servers in the proposed WS-Replication architecture. There would be essentially no difference in the way in which replicated webservices are invoked in comparison with their non-replicated counterparts. It is the responsibility of WS-Replication to deploy a webservice on

several servers, route calls to all these servers in a transparent fashion, receive replies from all of them and finally give one reply to the service consumer.

In [34], the authors propose to use redundant servers to handle failure of the primary web server. They propose an FT-SOAP system which contains four components to perform the following functions: replication management, fault management, logging/recovery mechanism and client FT transparency. The fault manager performs the tasks of fault detection and fault notification. The recovery manager captures and logs the invocation activities for recovery purpose. Replication manager takes up the responsibility of replicating webservices on redundant servers. These replicas are called as a fault-tolerant WS group.

For every fault-tolerant WS group, a new tag called Web Service Group (WSG), is appended to WSDL. This helps in achieving fault transparency at the end of the service consumer. In case of failure of the invoked service, this WSG can be examined by the SOAP engine deployed at service consumer. Using the WSG tag, SOAP engine at the end of the service consumer may attempt several times to invoke other replicas in order to successfully retrieve the service. In the case of not receiving a response from any of the servers listed in WSG, a failure exception is sent by the SOAP engine to the application requesting the service. In such a case, the client application should again search for another WSDL.

7 Conclusion

Checkpointing and recovery schemes designed for composite webservices should successfully recover the failed instances from faults so as to avoid SLA faults. This chapter discusses mainly three techniques for handling faults in composite webservices: checkpointing, substitution and redundant servers. Checkpointing is a technique that has proved itself time and again as a robust fault handling strategy across areas like distributed computing, operating systems, banking transactions, etc. Conventional checkpointing techniques include coordinated checkpointing, uncoordinated checkpointing, communication-induced checkpointing and log-based checkpointing and recovery. But conventional checkpointing schemes are not directly applicable to webservices because of their distinct characteristics. This chapter introduces four main webservice characteristics which have to be considered while taking checkpointing decisions. It also discusses various checkpointing and recovery schemes for composite webservices and presents a comparison of them in a tabular format.

Various substitution approaches that have been proposed for handling faults in webservices are also discussed. Substitution approach [4–7] works well when a callee fails: callee is replaced by a functionally similar service. The approach fails if the server hosting the caller itself fails. All the execution instances of the caller in the case of failure of the server hosting the caller would have to be re-executed from the beginning. This requires re-execution of all the failed instances of the caller resulting

in increased execution times. Checkpointing and recovery schemes can handle faults in both caller and called services, but involve considerable amount of rework.

Redundant servers help in quickly restoring failed instances webservices, but their maintenance poses considerable overhead. A variety of redundancy solutions proposed for handling web server failures are also discussed at the end.

References

1. Berson and Alex. *Master Data Management & Data Governance*. McGraw-Hill Education (India) Pvt Limited.
2. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.
3. K.S. May Chan, Judith Bishop, Johan Steyn, Luciano Baresi, and Sam Guinea. A fault taxonomy for web service composition. *Service-Oriented Computing Work- shops*, 4907:363–375, 2009.
4. An Liu, Li Qing, Liusheng Huang, and Mingjun Xiao. Facts: A framework for fault-tolerant composition of transactional web services. *IEEE Transactions on Services Computing*, 3(1):46–59, 2010.
5. R Ben Halima, Khalil Drira, and Mohamed Jmaiel. A qos-oriented reconfigurable middleware for self-healing web services. *IEEE International Conference on Web Service*, pages 104–111, 2008.
6. Onyeka Ezenwoye and S. Masoud Sadjadi. Trap/bpel: A framework for dynamic adaptation of composite services. *Proc of WEBIST*, pages 216–221, 2007.
7. H.E. Mansour and T. Dillon. Dependability and rollback recovery for composite web services. *IEEE Transactions on Services Computing*, 4(4):328–339, 2011.
8. Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. Automated application-level checkpointing of mpi programs. *Proc of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 84–94, 2003.
9. K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.
10. Elmootazbellah Nabil Elnozahy, Lorenzo Alvisi, and Yi-Min Wang. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.
11. Richard Koo and Sam Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, pages 23–31, 1987.
12. L. Lin and M. Ahamad. Checkpointing and rollback-recovery in distributed object based systems. *20th International Symposium Fault-Tolerant Computing*, pages 97–104, 1990.
13. Kamath Mohan and Krithi Ramamritham. Failure handling and coordinated execution of concurrent workflows. *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23–27, 1998*, pages 334–341, Aug 1998.
14. Manivannan Robert Netzer, D. Manivannan, Robert H. B. Netzer, and Mukesh Singhal. Finding consistent global checkpoints in a distributed computation. *IEEE Transactions on Parallel and Distributed Systems*, 8:623–627, 1997.
15. Robert HB Netzer and Jian Xu. Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel and distributed Systems*, 6(2):165–169, 1995.
16. Y. Robert, F. Vivien, and D. Zaidouni. On the complexity of scheduling checkpoints for computational workflows. *IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops*, pages 1–6, 2012.
17. LME Silva and Joao Gabriel Silva. Global checkpointing for distributed programs. *Proc of 11th Symposium on Reliable Distributed Systems*, pages 155–162, 1992.

18. Y. Tamir and C. H Sequin. Error recovery in multicomputers using global checkpoints. *Proc of the International Conference on Parallel Processing*, pages 32–41, 1984.
19. B. Bhargava and S.-R. Lian. Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach. *Seventh Symposium on Reliable Distributed Systems, 1988. Proc.*, pages 3–12, Oct 1988.
20. D.L Russell. State restoration in systems of communicating processes. *Proc of IEEE Transactions on Software Engineering*, 6(2):183–194, 1980.
21. Y.M WANG. Consistent global checkpoints that contain a set of local checkpoints. *Proc of IEEE Transactions on computers*, 46(4):456–468, 1997.
22. Lorenzo Alvisi, Karan Bhatia, and Keith Marzullo. Causality tracking in causal message-logging protocols. *Journal of Distributed Computing*, 15(1):1–15, 2002.
23. David B. Johnson and Willy Zwaenepoel. Recovery in distributed systems using optimistic message logging and check-pointing. *J. Algorithms*, 11(3):462–491, September 1990.
24. R. Strom and S. Yemini. Optimistic recovery in distributed systems. *Proc of IEEE Transactions on computers*, 3(3):20–226, 1985.
25. Willy Zwaenepoel and D.B. Johnson. Sender-Based Message Logging. *Proc of the Seventeenth International Symposium on Fault-Tolerant Computing*, pages 14–19, 1987.
26. Wesley Bland, Peng Du, Aurelien Bouteiller, Thomas Herault, and Bosilca. Extending the scope of the checkpoint-on-failure protocol for forward recovery in standard mpi. *Concurrency and Computation: Practice and Experience*, 25(17):2381–2393, 2013.
27. Soumaya Marzouk, Afef Jmal Maˆalej, and Mohamed Jmaiel. Aspect-oriented checkpointing approach of composed web services. *Proc of the 10th International Conference on Current Trends in Web Engineering*, pages 301–312, 2010.
28. Marta Rukoz, Yudith Cardinale, and Rafael Angarita. Faceta*: Checkpointing for transactional composite web service execution based on petri nets. *Procedia Computer Science*, 10:874–879, 2012.
29. Sagnika Sen, Haluk Demirkan, and Michael Goul. Towards a verifiable checkpointing scheme for agent-based inter-organizational workflow system" docking station" standards. *Proc of the 38th Annual Hawaii International Conference*, pages 165– 173, 2005.
30. Urban Susan D., Gao Le, Shrestha Rajiv, and Courter Andrew. Achieving recovery in service composition with assurance points and integration rules. *On the Move to Meaningful Internet Systems: OTM*, 6426:428–437, 2010.
31. Atluri Vani Vathsala and HrushikeshaMohanty. Interaction patterns based check- pointing of choreographed web services. *Proc of the 6th International Workshop on Principles of Engineering Service Oriented and Cloud Systems*, pages 28–37, 2014.
32. Atluri Vani Vathsala and HrushikeshaMohanty. Time and cost aware checkpointing of choreographed web services. *Proc of the 11th International Conference on Distributed Computing and Information Technology*, pages 207–219, 2015.
33. Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño Martínez, and Ricardo Jiménez-Peris. Ws-replication: A framework for highly available web services. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 357–366, New York, NY, USA, 2006. ACM.
34. Deron Liang, Chen-Liang Fang, Chyouhwa Chen, and Fengyi Lin. Fault tolerant web service. In *Software Engineering Conference, 2003. Tenth Asia-Pacific*, pages 310–319, Dec 2003.

Webservice Security

Ravi Kiran Kumar Meduri

Abstract Webservices have become quite common in enterprise solutions as organizations have started exposing their services and products to the outside world through their extranets and the internet. Since webservices offer a great deal of flexibility in implementing business processes that span across application systems built on heterogeneous technologies, adding end-to-end security to webservices has become very important to make them robust. As webservices communicate over HTTP through XML messages, we need security at the transport layer and at both the sending and receiving ends of the messages. This chapter details the concepts of webservice security at both the levels followed by different standards used for its implementation and concludes with a brief overview on Oracle Webservice Management Framework.

1 Introduction

With the emergence of Service-Oriented Computing (SOC) [1] which is based on service-oriented architecture, webservices have become a common mechanism to design, build, and consume software applications. In succinct terms, SOA is an architectural model that organizes software applications and infrastructure into a set of services that interact with one another in a loosely coupled and highly cohesive manner using standards-based and platform-independent protocols within a heterogeneous distributed environment. The key reason behind the emergence of SOA is the necessity to respond quickly to opportunities coupled with the need for business agility in the contemporary markets [2]. In order to accommodate these needs, enterprises have been trying to streamline their existing business processes besides exposing their in-house and various other packaged applications in a standardized manner, as webservices that can be composed to form highly cohesive and loosely coupled independent business processes.

R. K. K. Meduri (✉)
Department of Computer Science, Government Degree College, Gummalakshmipuram, Vizianagaram District, India
e-mail: ravikiranmeduri@gmail.com

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_6

As mentioned, webservices are the most promising choice to implement service-oriented architecture and achieve its strategic objectives which are essential to building modern software applications. Let us spend a moment to understand what a webservice is. In its essence, a webservice is a semantically well-defined abstraction of a set of computational activities that rely on a number of resources to meet the given customer needs or business requirements. In other words, a webservice just talks about what it does in an abstract way and does not even give a glimpse of how it performs its activities. This is the very reason why webservices have become platform independent making them the best choice for building applications that work over the internet. Though a webservice does not specify any internal details of its operational activities, its access over the internet makes it vulnerable to various security attacks. This chapter articulates various security threats associated with webservices in general and the countermeasures that address each of these threats along with a few case studies.

This chapter is organized into multiple sections and begins with a brief overview of webservices followed by their classification and interaction patterns. It then jumps to security, in general, followed by a brief explanation of security terminology used within this chapter and in general. It then elucidates key webservice threats [3] such as forged claims, loss of confidentiality, message modification, principal spoofing, loss of confidentiality, man in the middle, message replays and denial of service, and the countermeasures available both at transport layer level and message level to counterattack these threats. It also gives a brief idea of various security frameworks available to address some of these security threats and finally concludes with a case study on enabling webservice security using Oracle Webservice Management (OWSM).

1.1 Webservices Description

Webservices are defined using Web Service Description Language. This XML-based language abbreviated as WSDL defines what a webservice performs hiding all the implementation details which is why webservices provide platform-agnostic interfaces to integrate applications that run on heterogeneous platforms. The key elements of WSDL include:

- Types: to define the XML schema types that are used in the webservice definition.
- Message Types: to define the parametric types that can be used to pass messages to and from different operations.
- Port Types: to define the collection of operations that a webservice can perform along with the message types associated with each of its operations.
- Bindings: to specify the protocol to send the messages, the message style and the connection between the webservice operation and its actual execution point or the end point.
- Service: to specify the actual webservice execution URL or the end point URL.

So, in succinct terms, WSDL specifies what the webservice does and how it can be accessed without making any mention of its implementation details making it the best choice for integrating applications running on disparate technology stacks. Based on the definition of the WSDL, it can either be abstract (a blueprint without any implementation) or concrete (definition of an implemented webservice). Webservices are generally registered in a repository defined using UDDI framework. UDDI is an acronym for Universal Description, Discovery, and Integration. This framework is an XML-based, platform-agnostic and open specification for publishing and finding webservices. Based on their operational behavior, webservices can be classified as below.

1.2 Classification

As introduced, webservices are playing a key role in implementing today's software applications that operate over Internet. Before we go any further in understanding the other related aspects of webservices, it is worthwhile to have a look at different types of webservice provisioning techniques. Though the classification of webservices is not within the scope of this chapter, in order to understand the security framework depending on the provisioning technique, we need to have a quick look at the implementation variations. Based on the architectural style that is followed to implement webservices, we can classify them as Service-Oriented Architecture (SOA)-based webservices and Resource-Oriented Architecture (ROA)-based webservices. Services built using SOA use Simple Object Access Protocol (SOAP) while those built using ROA use Representational State Transfer (REST) framework. Let us have a quick juxtaposition between the two frameworks to understand the key differences that play an important role in identifying and implementing a security framework.

SOAP
SOAP [4] is a traditional framework suitable for implementing enterprise level Business-to-Business (B2B) solutions. It is a tightly coupled framework and generally consists of very large payloads consuming more bandwidth than its counterpart. SOAP always uses POST with complex XML request making response-caching difficult. SOAP is a transport-agnostic framework and is designed to implement distributed computing. It has an additional layer of security in terms of WS-Security and does not make any assumptions about the security available over the transport protocol it uses. Since it is predominantly used in building webservices, various standards and tools provide better support to this framework when compared to its counterparts.

REST
REST [1] is a newer technology as compared to its previous counterpart and associates itself with the implementation of a few mission-critical applications. It is a loosely coupled and lightweight framework utilizing less bandwidth for data transfers. REST can use GET making forward proxies and reverse proxies to cache the

responses. When a GET request is made through a forward proxy from a consumer, the response given by the corresponding provider can be cached within the forward proxy and any subsequent request can be directly responded through the cached message. Similarly, when an Internet-based consumer makes a request through a reverse proxy that is attached to a private network, the response obtained from a provider located within the private network can be cached and be used to service further requests of the same kind. REST works only on HTTP and is not a suitable candidate for distributed computing as it only supports point-to-point communication model. REST does not have any built-in support for security and relies on the transport layer security (HTTPS). Lack of support for standards, security and reliable messaging makes it a difficult choice to build sophisticated services.

1.3 Webservice Message Exchange Patterns

The general working nature of webservices includes a consumer who is interested to consume services offered by a provider which provides different services as a sequence of operations (Fig. 1).

During the process of interaction, a consumer and a provider together expose a pattern that demonstrates the way the two parties exchange messages. These are called Message Exchange Patterns (MEPs) [5]. The common MEPs are

- *Synchronous Request–Response Pattern*:
  This pattern makes a consumer wait until the corresponding provider processes the request received from the consumer and provides the response or a time-out occurs. Both request and response are exchanged in single connection.
- *Asynchronous Request–Response Pattern*:
  This is akin to synchronous pattern except that a consumer does not wait for the corresponding provider's response in the same connection.
- *One-way Request–Response Pattern*:
  This pattern is generally used for For Your Information (FYI) services. For this pattern, the response to a communication either does not exist or is not required.

Though these patterns provide the basic way of enabling interactions between a service provider and a service consumer, they are used in several permutations

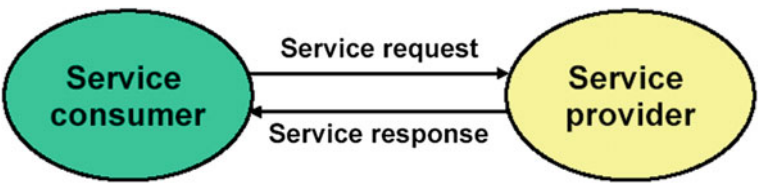


Fig. 1 Service-oriented interaction

and combinations among many webservices to provide greater functionality. These services are generally exposed to the external world through Internet and corporate extranets and therefore the security requirements of most of these webservices demand that the strongest possible safety mechanisms be implemented across their integration touch points. Before we jump into the implementation of webservice security frameworks, let us understand the security fundamentals for webservices in the next section.

2 Security Fundamentals

Since webservices are generally atomic in behavior, they are composed to implement the desired business functionality. The composed webservices require messages to be exchanged across multiple nodes (webservices) and webservices involved in this kind of message-based architecture span across heterogeneous environments. These scenarios indicate that security needs to be attached at the message level to support interoperability across heterogeneous platforms and message passing across multiple intermediary nodes.

Security in General:
Security [6] is very important as it is related to the protection of assets. Asset can be either tangible item such as a customer database or it can be a less tangible one such as reputation of a company.

We must understand that security is a chain and we need to find the weak links within the chain. In order to find the weak links with a view to improving the security, we need to identify the potential threats and the degree of risk that each threat presents. Once the threats and their corresponding risks are identified, we need to implement effective countermeasures. Therefore, it can be noted that security is about managing risks posed by different kinds of threats. Effective security can be achieved through an effective collaboration of people, process, and technology.

Security Elements:
Following are the key security elements on which all the security models rely on. Understanding these concepts is essential to provide end-to-end security to an IT application or a webservice.
Authentication: The Organization for the Advancement of Structured Information Standards (OASIS) [7] defines authentication as the corroboration that the peer entity is the actual one as claimed. This addresses the question of "who is accessing the application or service". This is a process of uniquely identifying the consumers of an IT application or a webservice.
Authorization: This addresses the question what a consumer can do after its authentication. It is the process of managing permissions to an authenticated consumer or entity or process so that it can access granted resources and perform allowed operations. Resources include databases, files, tables and so on while operations include transactions such a creating a purchase order, shipping an item to the end customer,

transferring funds from one account to another or changing the profile of a customer based on inputs.
Non-repudiation: Non-repudiation is a guarantee that a user cannot deny performing an operation or accessing a resource. This is essential to avoid fraudulent transactions and activities. Event logging and auditing are the keys to enable non-repudiation in any IT application or service.
Confidentiality: Confidentiality, also known as, privacy is the process of ensuring that data is accessible only to authorized entities. It prevents unauthorized users and eavesdroppers from gaining access to sensitive data illegally. Encryption and access control lists are commonly used to enforce confidentiality.
Integrity: Integrity is an assurance that data is protected from accidental or malicious modification while it passes across multiple networks. Hash techniques and message authentication codes provide integrity for data that goes across networks.
Availability: From a security viewpoint, availability ensures that an application or a service is available to legitimate users. As most of the security attacks aim at overwhelming an application or a service with huge data making it inaccessible to genuine user community, availability remains one of the key elements of security. The goal of availability is to protect an application or a service from Denial-of-Service (DoS) attacks.

All the security elements apply to any web application including those based on webservices. Therefore, it is fundamental to ensure that all these security elements are addressed during the design of a webservice to make it a secure webservice. To make a webservice secure, we first need to understand the possible security threats associated with webservices. The next sections give a brief idea of threat nomenclature followed by an overview of threats associated with webservices.

Security Issues—Terminology
When thinking about security, we have to understand the nomenclature associated with it so that we can describe the security policy in detail [8].

An ***asset*** is something that is worthy of being protected. Sensitive data, services, certain critical operations, and intellectual property are all assets. For instance, credit card numbers are an asset which needs to be protected within an application or while being sent over the network to a payment gateway. Updating customer rating is an example for critical operation that cannot be performed by all the legitimate users of the application and has to be available only to users with elevated privileges. A ***threat*** is a scenario wherein an information system is attacked through unauthorized access, modification of sensitive information and/or denial of service so that the activities, assets and/or individuals or an organization are adversely impacted. In short, a threat is a potential occurrence of an incident that damages an asset. Vulnerability is defined as a flaw in an information system, system security procedures, internal controls, or implementation that could be exploited. In common terminology, vulnerability is a weakness in a software component, giving rise to a threat. Vulnerabilities arise due to errors in configuration, inappropriate architecture/design or insecure coding techniques. For example, improper validation of input is a vulnerability that may cause input attacks at application layer. An ***attack*** is defined as an assault to a system

that is triggered by an intentional use of vulnerability. In other words, an attack enacts a threat by making use of security vulnerabilities.

In summary, a threat is a potential event that can adversely affect an asset while an attack exploits vulnerabilities in an application or a service to enact a threat.

Webservice Security Threats

As we have seen in security issue nomenclature, threat is the manifestation of a security vulnerability which can damage an asset. At this juncture, we need to understand various threats that are possible in the arena of webservices when accessed over Internet or through a corporate extranet. According to Webservices Interoperability (WS-I), following are the most important threats faced by webservices [9].

Intentionally modifying the original message and making the receiver consider it as the original message is called the ***message alteration***. This attack adds, deletes or modifies the data sent in a message to mislead the receiver. The disclosure of sensitive data to unauthorized entities, processes and/or individuals is termed as ***loss of confidentiality***. Obtaining a credit card number through an attack is an example for loss of confidentiality. Construction of fake messages with an intention to make the receiver believe that they are sent from a valid sender falls under attacks through ***falsified messages***. Relaying and possibly altering the communication between two parties and making them believe that they are directly communicating is popularly known as ***man-in-the-middle*** attack. This attack is enacted by a third party who sits between the sender and the receiver, and tries to modify messages making both participants unaware of the middleman. The attack wherein an attacker constructs and sends a message with credentials such that the message appears to be from a different authorized principal is called ***principal spoofing***. The attack of invoking a webservice through untrusted credentials comes under the category of ***forged claims***. In this attack, the attacker constructs and sends a message with false credentials that appear valid to the receiver. The attacks that send a previously sent message entirely or that include parts of a previously sent message in a new message are known as ***Replay (in parts and whole message)***. The motivation behind replaying the messages sent by original sender is to gain unauthorized access. The attack of forcing a webservice to perform large amounts of work through disproportionate payloads such that the valid requests are denied is called ***Denial-of-Service*** (DoS) attack.

3 Security Solutions, Mechanisms and Countermeasures

In order to counterattack the security threats explained in the previous section, several mechanisms have been proposed. In this section, let us have detailed a look at mechanisms to address these security challenges that can be applied to different communication layers. The primary layers where security can be provisioned are the transport layer and the SOAP messaging layer [10]. Both these layers can be configured on their own and/or in combination to address a variety of requirements.

Transport Layer Security

Webservices transmit SOAP messages over the transport layer. So the transport layer can act as a gateway to provide security to webservices by implementing various security measures. These security provisions provide integrity, confidentiality and authentication for HTTP messages that include the SOAP messages. These security mechanisms provide point-to-point security between a service consumer node and a service provider node. To understand the transport layer security provisions, we need to understand the concepts of Secure Socket Layer (SSL) and Transport Layer Security (TLS).

Secure Socket Layer and Transport Layer Security:

Secure Sockets Layer (SSL) is a security standard to establish an encrypted connection between a server and a client—typically a web server (website) and a browser. In the context of webservices, it establishes a secure link between a webservice provider and a webservice consumer. SSL aims at transmitting sensitive information such as login credentials, personal identifiers, banking transactions securely between a webservice provider and a webservice consumer. Normally, data is sent in plain text between a consumer and its provider—making the connection vulnerable to eavesdropping. If data is exchanged in plain text, an attacker can intercept all data (plain text) sent between a consumer and its provider and use it for attacking the entire system. SSL is a security protocol that encrypts the data being transmitted over the link established between a webservice provider and its consumer. SSL has been improvised over the years resolving many security issues. Since SSL version 3.0, it is renamed as Transport Layer Security (TLS). Though there are some minor differences between SSL and TLS protocols, they are not within the scope of this chapter and the readers are advised to go through the corresponding literature [11–13].

SSL is implemented in the form of a certificate. An SSL certificate is associated with a pair of keys out of which one is a public and the other is a private key. These keys are together used to establish an encrypted connection between a provider and a consumer. A certificate contains a "subject," which is the identity of the webservice provider. Once we get a certificate through a Certificate Signing Request (CSR), we need to install it on the consumer so that all the connections made to the provider are established securely.

When a consumer attempts to access a provider running on an HTTP server over SSL, the consumer and the provider establish a secure connection through "SSL Handshake" [13] that happens in the background without the user having any idea on the internal details of the connection establishment. The handshake process uses three keys—public, private, and session keys while setting up an SSL connection between a provider and a consumer. Though public and private keys alone can be used for encrypting and decrypting messages, the asymmetric key infrastructure usually consumes more processing power than symmetric process. Therefore, these keys are only used to generate a symmetric key while setting up the SLL connection. After the establishment of SSL connection between the provider and the consumer, messages will be encrypted and decrypted using the symmetric session key. The

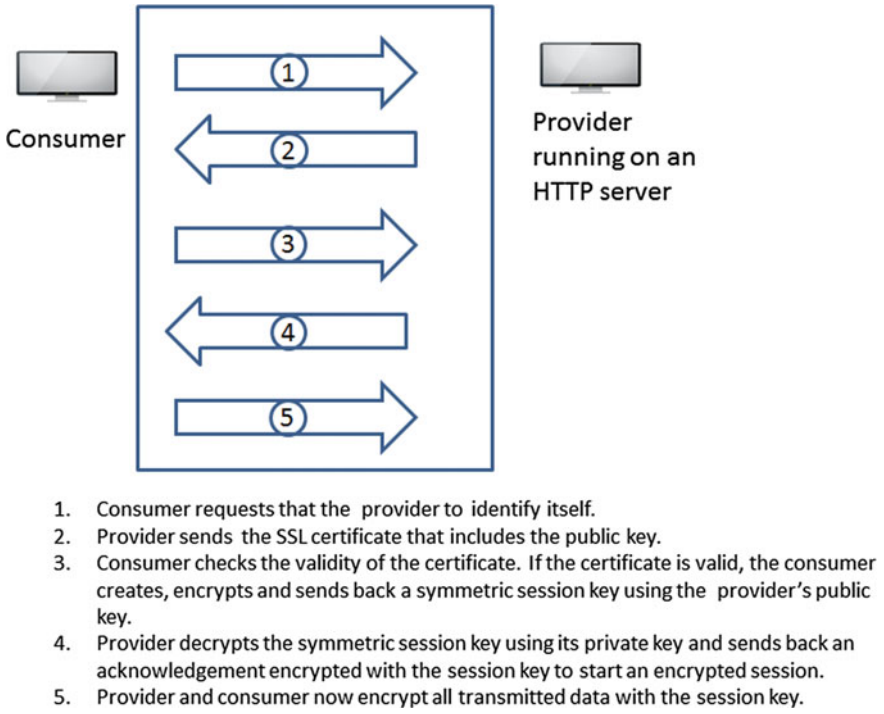


Fig. 2 SSL Handshake

following diagram explains of SSL Handshake which is also popularly known as Three-Way SSL Handshake (Fig. 2).

Let us now understand how SSL/TLS provides the basic authentication, integrity, and confidentiality for SOAP messages.

Confidentiality

SSL/TLS, when used in association with HTTP (HTTPS), provides confidentiality for messages exchanged between a provider and a consumer. During SSL handshake, a consumer and a provider determine the encryption algorithm and the symmetric key to be used for the duration of the session. Both the consumer and the provider use the agreed-upon encryption algorithm and the session key to encrypt messages ensuring that the data exchanged remains a secret even if the session is intercepted by unauthorized entities. Since SSL/TLS uses asymmetric encryption to transport the session key between a provider and a consumer during the SSL session establishment, the same session key cannot be known to other sessions thus avoiding the possibility of an attack on data privacy. However, it should be noted that SSL/TLS provides confidentiality only for the duration of an active HTTPS session. It does not provide any protection for messages that are already received. We also have to note that SSL/TLS provides confidentiality to a complete message. It cannot be used to add privacy to parts of a message making SSL/TLS inappropriate in scenarios where a

message passes through several intermediate nodes each having its own requirement to secure specific parts of the message.

Integrity
SSL/TLS, when used in combination with HTTP (HTTPS), it provides integrity to an HTTP message which in turn includes the actual data to be exchanged. It uses secure hash functions such as SHA [14, 15] and MD5 [14] to calculate message digests or Message Authentication Codes (MAC) [16]. These MACs are sent along with the actual messages to provide integrity. As in the case of confidentiality, integrity is also provided only for the duration of the HTTP session. Once a message is delivered through HTTPS to a consumer or a provider, message integrity will cease to exist. This means end-to-end message integrity cannot be provided using SSL/TLS. Another shortcoming of communication over HTTPS is that SSL/TLS provides integrity to the entire message and cannot digitally sign parts of a message thus making it unsuitable for providing end-to-end integrity for a message passing across a topology of webservices.

Server Authentication
The SSL/TLS in its basic form provides authentication of a provider to its consumer or a webservice client. Once the consumer generates a symmetric key, it encrypts and sends it using the public key of the provider. The provider can identify the symmetric key only if it can decrypt the symmetric key using the right private key. If the provider does not have the correct private key, the authentication of the server fails.

Consumer Authentication
A service provider can authenticate its consumer provided it requests the later to share its certificate (client certificate). The client certificate typically is X.509 certificate [17]. Once the SSL/TLS handshake is performed between a provider and its consumer, the provider validates the correctness of the certificate shared by the consumer. The provider responds back to the consumer with a "finished" message only if the client certificate is found to be from a trusted Certifying Authority (CA) and is valid. This mechanism can be considered as SSL/TLS handshake with Client Certificates and is depicted in Fig. 3.

HTTP Authentication
Authentication of a consumer over HTTP can be provided without SSL/TLS as well. This method is very popular with web applications but is vulnerable to security attacks. The same, however, can be used for authenticating webservice consumers as well. This mechanism includes two types of authentication methods:

1. *Basic Authentication* [18] uses the credentials to be sent in HTTP authorization header in plaintext. Both username and password are concatenated using a ":" and the concatenated string is sent using Base64 encoding scheme. Since the credentials are not encrypted, this method is not recommended over HTTP but can be used over HTTPS (SSL/TLS) as the credentials are encrypted.
2. *Digest Authentication* [18] alleviates the problem of sending credentials in Base64 encoding format and incorporates digest mechanism to send the credentials over HTTP. This method uses a hash function to calculate the digest

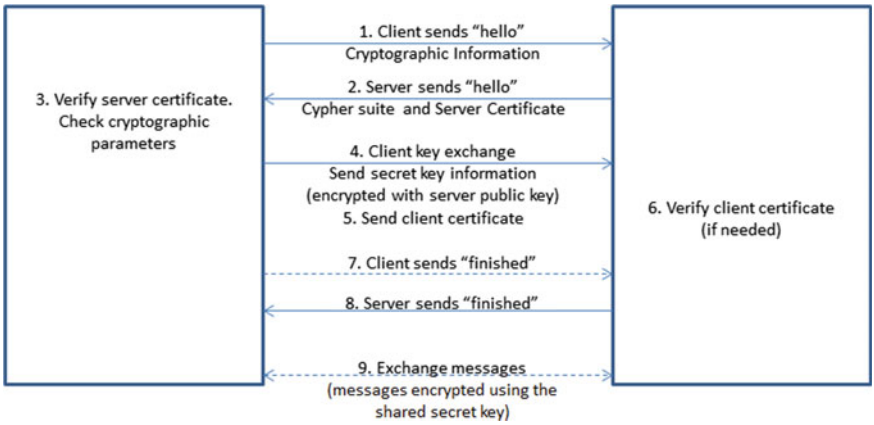


Fig. 3 SSL/TLS Handshake with client certificates

based on the username, password, a unique value generated by the client known as "nonce", security realm, the request count, the Quality of Protection (QoP), and the URL of the resource being requested. The hash value along with all the other values except the password are sent in the HTTP header. The server then uses all the values to compute the hash and checks if it matches with the one received in the HTTP header. If the hash values match, then the consumer's request is serviced. This method is more secure than the basic authentication as the credentials are not sent in plain text. When used with HTTPS, digest authentication mechanism makes webservices more secure. More details on the use of HTTP authentication scheme are available in RFC 2617 [18]. Readers are advised to go through it before using it for securing webservices.

Message-Level Security

Though transport layer security provides authentication, integrity and confidentiality for messages exchanged between a consumer and its provider, it needs to be enabled between each pair of nodes when webservices are orchestrated or choreographed in a Service-Oriented Architecture. In other words, transport layer security provides point-to-point security and needs to be extended across all pairs of nodes to enable end-to-end security when a message passes across multiple webservices. Enforcing transport layer security at each node may not be possible. Therefore, plugging in the security features within the message will be a better option when compared to the former. The mechanism used to attach security to SOAP messages is popularly known as "SOAP Message Security" and this section gives various details of enriching SOAP messages with security.

Message-level security is provided at the application layer and it ensures the protection of messages exchanged between applications. Therefore, message-level security is essential.

- When messages are exchanged asynchronously.
- When security at the application level is important.
- When messages flow across multiple nodes connected with different protocols.

Since message-level security is provided at the application level which is the highest layer in a network stack, security at the bottom layers becomes needless. Webservices built using SOAP often use the message-level security while the REST services typically rely on transport layer security.

Message-level security attaches security information to a SOAP message so that the security information travels along with the message. This feature provides end-to-end security for messages that are sent across a topology of webservices. For instance, let us consider a scenario where a part of a SOAP message is signed and encrypted by a sender for a particular receiver. If the message passes through a network of services before arriving at its intended recipient, the digitally signed and encrypted parts cannot be visible to any of the intermediate services and can only be visible to the actual recipient. In other words, message-level security achieves end-to-end security by providing security from the origination of a message until it reaches its destination.

Some of the advantages of message-layer security include the following:

- Security information travels along with the message from its origination until it reaches the destination providing end-to-end security.
- Specific parts of a message and attachments can be secured as well.
- Security information can be applied even at intermediate services when messages are exchanged across a network of services.
- Security provided at the message level is neither dependent on a specific application nor dependent on a specific protocol.

The drawback of message-layer security is that it is relatively more complex as compared to transport layer security and adds overhead to message processing. It becomes more difficult to handle message-level security when a message passes through several intermediaries with different security requirements.

Message-level security is provided by several policy mechanisms and we need to understand them before we see how they provide various nuances of security discussed in the earlier sections of this chapter.

WS-Security

SOAP message security is provided by "**Web Services—Security (WS-Security, WSS)**" which is an extension to SOAP and is published by **Organization for the Advancement of Structured Information Standards (OASIS)** [7]. WS-Security defines three mechanisms to enable security for the messages exchanged using SOAP across multiple webservices:

- *XML Digital Signature*: Signing SOAP messages to ensure integrity and provide non-repudiation.
- *XML Encryption*: Encrypting SOAP messages to assure confidentiality.
- *WS-Security Tokens*: Attaching security tokens such as X.509 certificates [19, 20], Kerberos tickets [21, 22], User ID/Password credentials, Security Assertion Markup Language (SAML) [23] Assertions and custom-defined tokens to ratify the identity of the sender.

WSS attaches the aforementioned security structures in the header section of a SOAP message and works in the application layer providing end-to-end security. However, WSS is just a specification and does not secure webservices on its own. This specification, when used in combination with webservice frameworks such as Oracle Webservice Management (OWSM) and higher level application-specific protocols, creates a secure channel between a provider and a consumer. The standards offered by this specification enforce a wide gamut of security features when implemented with various application-level frameworks. While using any of these frameworks along with WSS specification, it is the responsibility of the implementer to make sure that the connections established are not vulnerable.

XML Digital Signature

Digital signatures use Public Key Cryptography (PKC) [24, 25], which is based on an algorithm that uses two different but mathematically related keys—one to create the digital signature of a message to be secured and the other to verify the digital signature. This asymmetric key mechanism uses the private key of the sender to encrypt a message, whereas the encrypted message can only be decrypted using the public (available to all) key of the sender of the message. This key pair security mechanism is considered to be the strongest available security mechanism as it is computationally impossible to deduce one key from the other. This very feature of PKC provides the following security measures:

A consumer can identify the identity of its provider as it can decrypt an inbound message using its private only if it is sent after encryption by the corresponding public key. This feature enables webservice provider *authentication*. If a message is encrypted by the public key of a consumer, it can only be viewed after decrypting it with the corresponding paired private key. This mechanism achieves message *confidentiality*. If there is any tampering of a message sent by a provider to a consumer during its transit, it can easily be identified by comparing message digest values thereby achieving *message integrity*. PKC in combination with message hashing encures non-repudiation as a provider cannot deny sending a message or any actions committed.

Let us not understand how a digital signature can be created. The following sequence of steps illustrates the creation of a digital signature:

Step1: The sender generates the message digest for a given message using a message digest algorithm such as SHA [11]. The message digest algorithms are so robust that even a change in a single character of the original message will result in a different message digest. This mechanism, hence, ensures message integrity.
Step2: The sender encrypts the message digest generated in Step1 using the private key. This is called the digital signature of the message. The encrypted message can only be decrypted using the corresponding public key ensuring sender authentication.
Step3: The sender attaches the digital signature to the original message and sends it to the receiver.

The digital signature protocol helps to ensure the following while being used in webservices:

- A signature ensures the authenticity of the provider.
- A signature cannot be copied as a private key is unique and is known only to the provider.
- Two different messages do not produce the same digital signature as it is defined as a function of the message and is unique for each message.
- A digitally signed message cannot be altered during transit as any unidentified alteration will make the signature verification fail at the receiver's end since the regenerated hash value will be different from the actual hash value.
- A digital signature cannot be repudiated as the message and the corresponding signature can be changed only by the provider through the private key.

The process of digitally signing a message is depicted in Fig. 4 [26].

The above fundamentals lay the basis to create XML digital signatures which are specifically meant for exchanging XML data over the Internet. XML digital signatures are digital signatures designed for use by XML transactions. These digital signatures are used to protect webservices by adding authentication, integrity and support for non-repudiation. An important fundamental feature of XML digital signature is that it can be applied to specific portions of the XML content. This flexibility is important when XML document has a long history where different parties add different contents at different instances of time as each party needs to sign the content relevant to itself without worrying about the content signed by other parties. This feature is also important in cases where we want to maintain the integrity of only some parts of an XML document while leaving other parts to change. This option is

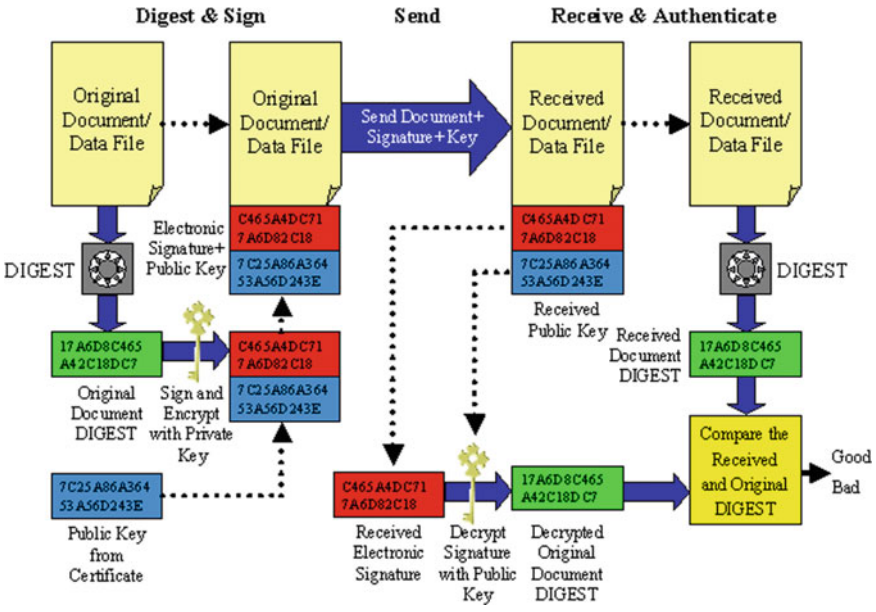


Fig. 4 Digital signature validation

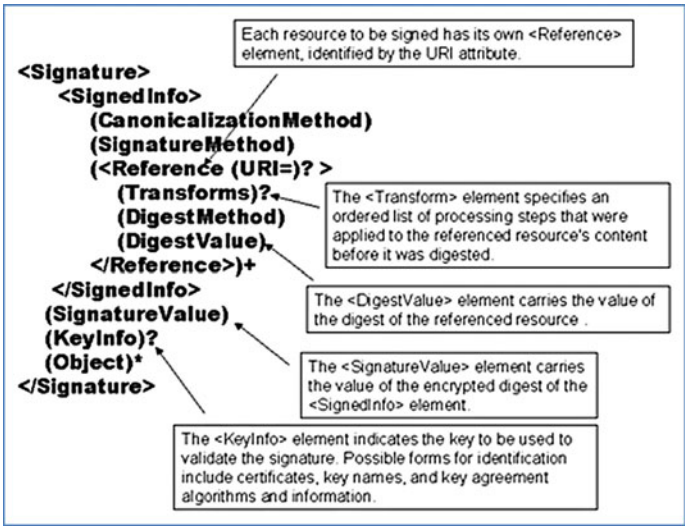


Fig. 5 Components of an XML digital signature [25]

useful when users are allowed to make changes to a signed XML message. In this case, only the non-modifiable content can be digitally signed while leaving the other changeable portions of the message unsigned.

Signature validation mandates the accessibility of the data object that is being signed. The location of the object being signed is specified in the XML signature itself. The object being signed can be referenced by an URI within the XML signature or can reside within the same resource as the XML signature. The XML signature and the original object being signed can exist in a sibling relationship (the signature is a sibling to the original object) or in parent–child relationship (the signature is the parent of the original object) or in child–parent relationship (the signature is the child of the original object) with respect to each other. In other words, both the object being signed and its digital signature can reside in any hierarchy within an XML message. However, we must ensure that they are accessible within a given XML message. Figure 5 [25] shows the basic components of the XML digital signature:

Let us now have a quick glance at the creation of an XML digital signature followed by its validation procedure.

XML Digital Signature Creation:

1. Determine the resource or the portion of the XML content to be signed.
2. Canonicalize and calculate the digest for each of the identified resources.
3. Collect all the reference elements along with their digests under the <SignedInfo> element.
4. Canonicalize and calculate the digest for the <SignedInfo> element.
5. Encrypt the digest value of the <SignedInfo> element using a key.
6. Add the key details used for encryption in the XML digital signature message.

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo Id="foobar">
  <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
  <Reference URI="http://www.abccompany.com/news/2000/03_27_00.htm">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
   <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
  </Reference>
  <Reference URI="http://www.w3.org/TR/2000/WD-xmldsig-core-20000228/signature-example.xml">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
  </Reference>
 </SignedInfo>
 <SignatureValue>MC0E~LE=</SignatureValue>
 <KeyInfo>
  <X509Data>
   <X509SubjectName>CN=Ed Simon,O=XMLSec Inc.,ST=OTTAWA,C=CA</X509SubjectName>
   <X509Certificate>MIID5jCCA0+gA...lVN</X509Certificate>
  </X509Data>
 </KeyInfo>
</Signature>
```

Fig. 6 A digitally signed message

Figure 6 shows an example message [25] that is digitally signed using SHA1 [15] for generating the message digest and X.509 certificate [17] for its encryption.

XML Digital Signature Validation:
As illustrated in Fig. 4, the signature validation process involves three major steps. First, the digest of the original document received is calculated. This is done by calculating the digest of the *SignedInfo* element (as shown in Fig. 6) with the help of the digest algorithm mentioned in *SignatureMethod* element. The second step decrypts the signed digest value given in SignatureValue element using the public key sent in *KeyInfo* element to get the original digest value calculated by the sender. In the third step, both the digest values are compared and if both are the same, the signature validation becomes successful. Otherwise, it indicates the original message was tampered in transit. In essence, XML digital signatures allow us to enable data integrity for messages exchanged in the form XML either partially or fully.

XML Encryption
In order to provide confidentiality to messages exchanged across a webservice topology, we need a mechanism to encrypt the data. This feature is provided by XML encryption standard [27]. One of the most striking features of XML encryption is to enable the secure exchange of a message between endpoints of webservices that coordinate with multiple intermediaries. This feature guarantees end-to-end security, unlike transport layer encryption which encrypts data exchanged between entities connected in a point-to-point manner. XML encryption takes place at the application layer encapsulating the encrypted message in the actual message to be exchanged.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <getCompanyInfo xmlns="http://www.oracle.com">
   <name>Company</name>
   <description>XML Security Company</description>
  </getCompanyInfo>
 </s:Body>
</s:Envelope>
```

Fig. 7 Unencrypted message

The application layer encryption also ensures that the encrypted data remains opaque to all the intermediaries and can be decrypted only at the destination service.

Before we get into various ways used for encrypting XML content, let us check an example message and its encrypted content. Figure 7 shows an unencrypted message while Fig. 8 shows the encrypted version of the former.

Unencrypted SOAP Message:
See Fig. 7.

Encrypted SOAP Message:
See Fig. 8.

Let us understand the terminology used in the encrypted message [28]:

Encrypted Key: The *EncryptedKey* element details out the information related to the encryption key.
Encryption Method: The *EncryptionMethod* element specifies the algorithm in "*Algorithm*" attribute and the size of the key used for encryption in "*KeySize*" attribute. For instance, Fig. 8 shows an encrypted message which carries the original message given in Fig. 7. The original message is encrypted by means of the Advanced Encryption Standard (AES) [29] symmetric cipher, while the session key (*EncryptedKey*) encryption uses the RSA [30] asymmetric algorithm.
Cipher Value: The *CipherValue* element holds the data that is encrypted using the encrypted key and the encryption algorithm.
Key Info: The *KeyInfo* element carries details of the key used in encryption along with information about the recipient. For instance, the message given in Fig. 8 shows a key that bundles an X.509 certificate and its common name.
Reference List: The *ReferenceList* element lists out a set of references to the encrypted parts of a message. Each *Data Reference* points to the message encrypted and enclosed in *Encrypted Data* element.
Encrypted Data: The *EncryptedData* element contains the message that is encrypted. In Fig. 8, the entire SOAP Body element is replaced by *EncryptedData* as SOAP body is encrypted.

The XML encryption mechanism, essentially, is similar to the approach used in the transport layer encryption mechanism with the exception that parts of the XML

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext" s:actor="Enc">
  <!-- Encapsulates the recipient's key details -->
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#" Id="00004190E5D1-7529AA14"
MimeType="text/xml">
    <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#rsa-1_5">
     <enc:KeySize>256</enc:KeySize>
    </enc:EncryptionMethod>
    <enc:CipherData> <!-- The session key encrypted with the recipient's public key -->
     <enc:CipherValue> AAAAAJ/IK ... mrTF8Egg== </enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>sample</dsig:KeyName>
     <dsig:X509Data> <!-- The recipient's X.509 certificate -->
      <dsig:X509Certificate> MIIEZzCCA0 ... fzmc/YR5gA </dsig:X509Certificate>
     </dsig:X509Data>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
     <enc:DataReference URI="#00004190E5D1-5F889C11"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
            Id="00004190E5D1-5F889C11" MimeType="text/xml"
                              Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData> <!-- The SOAP Body encrypted with the session key -->
   <enc:CipherValue> E2ioF8ib2r ... KJAnrX0GQV</enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
<s:Envelope>
```

Fig. 8 Encrypted message

content can be encrypted whereas the SSL/TLS encrypts the entire XML content. The following steps give a bird's eye view of the encryption process:

1. The provider generates a session specific *symmetric* key which is used for encrypting messages exchanged between the provider and its consumer. This symmetric key is succinctly termed as a session key.
2. The session key is in turn encrypted with the public key of the consumer. This step ensures that no other entity can extract the session key even if it gets access to the message because it can only be decrypted by the private key of the intended consumer.
3. Once session key is generated and sent to the consumer, the provider starts sending data encrypted using the session key.

4. The provider sends a message after encrypting it with the session key along with the session key encrypted with the consumer's public key.
5. The consumer upon receiving a message extracts and decrypts the session key using its *private key*.
6. The consumer then decrypts the encrypted data using the session key to get the actual plaintext message.

The fundamental benefit of XML encryption is to provide confidentiality to messages exchanged between different end points besides equipping the sender and the receiver to encrypt only parts of the message that needs to be confidential leaving the rest of the message to be sent in plain text. The only overhead with this approach is that when an encrypted message needs to be shared with multiple recipients, it needs to include the sender's key multiple times as it needs to be encrypted with the public keys of each of the intended recipients.

Webservice Security Tokens

Authentication is the first step to enable security while accessing any resource as it ratifies the identity of a sender. Without validating sender's identity, there is no point in trying to check the integrity of the message sent by the sender. Therefore, validating the authenticity of the sender of a message is the first step to enable webservice security.

In Webservice Security (WSS), authentication is provided by three kinds of tokens—Username Token, Binary Security Tokens, and XML-based Security Tokens. X.509 Token and Kerberos Token are grouped under binary security tokens while Security Assertion Markup Language (SAML) Token and Rights Expression Language (REL) Token provide means to send XML-based security tokens. These authentication policies enable a webservice consumer to send its credentials that will be ratified by its provider against the identity store and the identification mechanism defined by the provider. These security tokens travel across different webservice endpoints in a webservice topology in the header section of a SOAP message so that the sender of the message can be identified at any time during the message transit across different nodes (services). The following sections give an exploration of these security tokens and their usage within SOAP framework.

Username Token

According to OASIS [7], a Username Token [31] is used by a webservice consumer to validate its identity while invoking a webservice provider. Username Token includes "Username" as a mandatory element. It optionally includes "Password" either in the form of plain text or in the form of a password digest specified as SHA1 [15] hash value. Two more optional values—"Nonce" and "Created" are introduced in the specification of the token to provide a countermeasure for replay attacks.

Figure 9 presents a sample SOAP message that makes use of Username Token element. This example, taken from OASIS username token reference [31], sends the password in clear text and hence mandates that the message be sent through a secure channel such as HTTPS:

Similarly, Fig. 10 describes a SOAP message that uses password digest in combination with a nonce and timestamp [31]:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Zoe</wsse:Username>
        <wsse:Password>IloveDogs</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S11:Header>
</S11:Envelope>
```

Fig. 9 Username token with a plaintext password

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu= "...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>NNK</wsse:Username>
        <wsse:Password Type="...#PasswordDigest">weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==</wsse:Password>
        <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
        <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S11:Header>
  ...
</S11:Envelope>
```

Fig. 10 Username token with a digest password

A useful extension to the username token is to use the password along with the optional "Salt" and "Iteration" values to derive the secret key that can be used to provide integrity and confidentiality to messages through Message Authentication Codes (MAC) or encryption. Readers can refer to the OASIS standard specification [31] for Username Token for the implementation of key derivation using the password.

Apart from enforcing authentication, username token addresses replay attacks through the use of timestamp, nonce, caching, and any other application-specific security tracking mechanisms. However, transport layer security needs to be enabled to attach confidentiality and integrity to username token. If the transport layer is not secure, the password must be digested besides being kept strong enough to thwart password guessing attacks.

X.509 Token

X.509 security profile [19, 20] is another authentication mechanism that is based on X.509 certificate framework to provide security to webservices. An X.509 certificate

describes the binding between a webservice and a public key in the form of a set of attributes. These attributes include the owner of the certificate, the issuer of the certificate, the serial number of the certificate, and the period during which the certificate is valid. This binding may be revoked by mechanisms such as issuance of CRLs (Certification Revocation Lists), OCSP (Open Certificate Status Protocol) tokens or provisions that are external to the X.509 framework. An X.509 certificate is generally used to authenticate the sender of a SOAP message by validating its public key. It can also be used to identify the public key associated with an encrypted SOAP message.

The X.509 security token type can be a single X.509 certificate, a sequential list of X.509 certificates bundled in PKIPath (Public Key Infrastructure Path), or a list of X.509 certificates along with an optional list of CRLs packaged in PKCS #7 [32] (Public Key Cryptography Standard). The X.509 security tokens can be referenced using a Key Identifier or as a Binary Security Token or by specifying the certificate Issuer Name the certificate Serial Number.

Figure 11 [19] shows a single X.509 certificate embedded in a SOAP message as a binary security token.

Figure 12 [19] shows a SOAP message digitally signed with the help of an X.509 token referenced by means of a *KeyIdentifier*. In this message, the signature is embedded in *SignedInfo element* that includes the actual message body and the certificate used for its signing. Message body is referenced using the identifier #body to the message body while its signing certificate is referenced through the reference #keyinfo to the *KeyInfo* element. The *KeyInfo* element in this example contains a reference to an X.509 certificate but not the actual certificate itself. Therefore, it uses a transformation to replace the reference to the certificate with the original certificate. The *KeyInfo* element specifies X.509 certificate using its subject key identifier.

Similarly, Fig. 13 [19] shows a sample SOAP message digitally signed using an X.509 certificate referenced by means of its issuer name and its serial number. The signature of the message is included in the *SignedInfo* element which contains references to the actual message (#body) and the key (#keyinfo) used for signing. The key is referenced by means of its issuer and serial number which are mentioned in *X509IssuerSerial* element in Base64 format inside *SecurityTokenReference*.

X.509 security tokens can also be used to provide integrity through digital signatures and confidentiality through encryption. They can also be used at the transport layer to enable SSL/TLS encryption.

Kerberos Token

Kerberos token is an additional grant from the authentication system to access the desired network resources. Once a consumer authenticates itself using a username token or an X.509 security token, the security system presents a ticket granting token (TGT). The TGT is an opaque piece of data that a consumer cannot read but has to provide to get a service ticket (ST) which in turn provides access to the desired resources.

The following sequence of steps [33] illustrates the Kerberos protocol mechanism to access a webservice provider:

```
<S11:Envelope xmlns:S11="...">
  <S11:Header>
    <wsse:Security
       xmlns:wsse="..."
       xmlns:wsu="...">
      <wsse:BinarySecurityToken
         wsu:Id="binarytoken"
         ValueType="wsse:X509v3"
         EncodingType="wsse:Base64Binary">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0...
      </wsse:BinarySecurityToken>
      <ds:Signature
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>...
          <ds:Reference URI="#body">...</ds:Reference>
          <ds:Reference URI="#binarytoken">...</ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>HFLP...</ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#binarytoken" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="body"
     xmlns:wsu="...">
    ...
  </S11:Body>
</S11:Envelope>
```

Fig. 11 X.509 certificate as a binary security token

1. A consumer sends an AS-REQ packet to authenticate itself against a Key Distribution Center (KDC). The AS-REQ packet carries a username security token or an X.509 security token.
2. The KDC, after validating the AS-REQ packet, sends a TGT in AS-REP response packet. The TGT is an opaque token which the consumer cannot read but can use for accessing resources such as a webservice provider.
3. The client uses the TGT and sends a TG-REQ with the TGT to the Ticket Granting Service (TGS) to obtain a Service Ticket (ST) to access a specific webservice provider and/or its operations.
4. The TGS responds with a TG-REP response that consists of the ST.

```
<S11:Envelope xmlns:S11="...">
  <S11:Header>
    <wsse:Security
       xmlns:wsse="..."
       xmlns:wsu="...">
      <ds:Signature
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>...
          <ds:Reference URI="#body">...</ds:Reference>
          <ds:Reference URI="#keyinfo">
            <ds:Transforms>
              <ds:Transform  Algorithm="...#STR-Transform">
                <wsse:TransformationParameters>
                  <ds:CanonicalizationMethod Algorithm="..."/>
                </wsse:TransformationParameters>
              </ds:Transform>
            </ds:Transforms>...
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>HFLP...</ds:SignatureValue>
        <ds:KeyInfo Id="keyinfo">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="...#Base64Binary"
                ValueType="...#X509SubjectKeyIdentifier">
              MIGfMa0GCSq...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="body"
     xmlns:wsu=".../">
    ...
  </S11:Body>
</S11:Envelope>
```

Fig. 12 X.509 certificate as a key identifier

5. The consumer then sends the ST in an AP-REQ packet to access the required network resource.
6. The webservice provider responds back with AP-REP response with successful authorization.

Figure 14 diagram illustrates the Kerberos authentication process among service consumer, identity provider and service provider:

```
<S11:Envelope xmlns:S11="...">
  <S11:Header>
    <wsse:Security
       xmlns:wsse="..."
       xmlns:wsu="...">
      <ds:Signature
          xmlns:ds="...">
        <ds:SignedInfo>...
          <ds:Reference URI="#body"></ds:Reference>
          <ds:Reference URI="#keyinfo"></ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>HFLP...</ds:SignatureValue>
        <ds:KeyInfo Id="keyinfo">
          <wsse:SecurityTokenReference>
            <ds:X509Data>
              <ds:X509IssuerSerial>
                <ds:X509IssuerName>
                  DC=ACMECorp, DC=com
                </ds:X509IssuerName>
                <ds:X509SerialNumber>12345678</X509SerialNumber>
              </ds:X509IssuerSerial>
            </ds:X509Data>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="body"
     xmlns:wsu="...">
    ...
  </S11:Body>
</S11:Envelope>
```

Fig. 13 X.509 certificate specification using an issuer and a serial

Coming to the provisioning of authentication using a Kerberos token, a consumer uses WS-Security framework to include a Kerberos token in a SOAP request message to obtain access to its provider. However, WS-Security specification is limited to using only AP-REQ packet (Service Ticket + Authenticator). The process to obtain a Service Token (ST) is not specified in WS-Security and should be implemented separately using Kerberos frameworks [34]. The implementation process to obtain an AP-REQ to send authentication tokens using WS-Security is out of scope for this chapter.

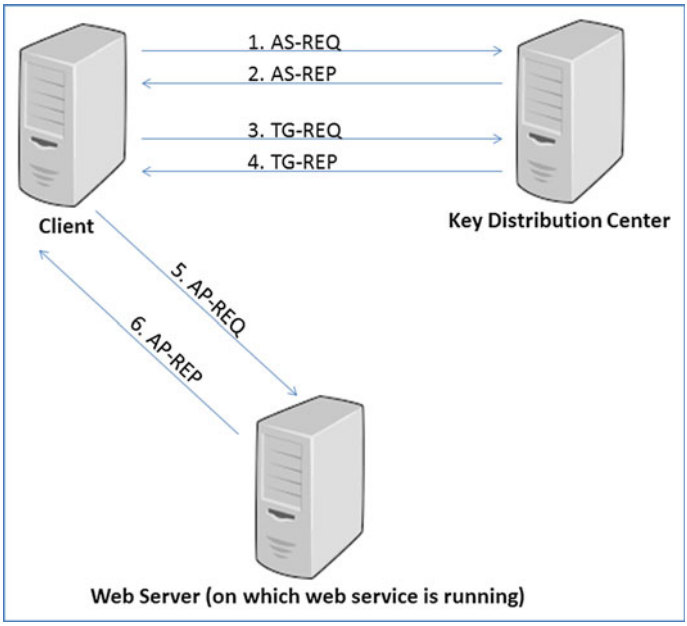


Fig. 14 Kerberos authentication process

```
<S11:Envelope xmlns:S11="...">
  <S11:Header>
    <wsse:Security xmlns:wsse="...">
      <wsse:BinarySecurityToken xmlns:wsse="... "
                    wsu:Id="myToken"
                    ValueType="...#Kerberosv5_AP_REQ"
                    EncodingType="...#Base64Binary">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0...
      </wsse:BinarySecurityToken>
      ...
    </wsse:Security>
  </S11:Header>
  <S11:Body>
    ...
  </S11:Body>
</S11:Envelope>
```

Fig. 15 A sample Kerberos security token

A Kerberos security token can be specified as binary security token encoded using an encoding scheme. Figure 15 shows a sample Kerberos security token encoded using Base64 encoding scheme:

Kerberos token can be used to provide authentication when used in digital signatures. When used as an encryption key, along with a symmetric encryption algorithm, it can be used to encrypt a message.

One potential threat with Kerberos tokens is their reuse which can result in replay attacks. In order to mitigate this threat, we can use timestamps, caching and application-specific message tracking mechanisms. Kerberos tokens cannot provide protection against message alteration and eavesdropping. These threats can, however, be mitigated with the help of confidentiality and integrity provisions of WS-Security framework.

SAML Security Token

The Security Assertion Markup Language (SAML) [35] is an open framework for sharing security information over the Internet through XML documents. The security information is expressed in terms of statements called assertions. SAML has become more popular than other security frameworks over the years due to the benefits it offers. The SAML framework is useful [36] to maintain cookies across multiple Internet domains, to enable single sign-on (SSO), to define security tokens in various webservices security frameworks and to enforce security in various phases of a business transaction.

SAML framework includes four basic parts [36]:

1. Assertions: to specify how the information related to the identification and access is defined.
2. Protocols: to describe how SAML Request and Response can be used to retrieve the assertions needed.
3. Bindings: to specify how SAML protocol can work in the transport layer and messages layer.
4. Profiles: to specify how SAML Protocols and Bindings can together be used to defend various security attacks.

In the webservice security framework, we only specify SAML assertions while protocols and bindings are automatically enforced by the framework itself. Apart from the aforesaid benefits, SAML has become very popular in webservice security because of the expressive nature of SAML assertions and their ability to thwart replay and man-in-the-middle attacks.

The assertions of SAML can be categorized into three types [36]. An *authentication statement* is used to identify a subject at any given instance of time. An *attribute statement* issued by an attribute authority declares the values of attributes associated with a subject based on defined policies. An *authorization decision statement* issued by an authorization authority declares if a subject can be granted access to perform an action on a given resource.

WS-Security provides three different methods to confirm the subject (the sender) of the message using SAML assertions. *Sender-Vouches* enables a third party attesting authority to vouch for the authentication of a subject. This method, however, needs the recipient to establish a trust relationship with the attesting authority. It should also be noted that the attesting authority is responsible to protect the actual message

```
<saml2:Assertion xmlns:saml2="..." xmlns:ds="..." xmlns:xsi="...">
  <saml2:Subject>
    <saml2:NameID>...</saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches">
      <saml2:SubjectConfirmationData>Address="129.148.9.42"</saml2:SubjectConfirmationData>
    </saml2:SubjectConfirmation>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
      <saml2:SubjectConfirmationData xsi:type="saml2:KeyInfoConfirmationDataType">
        <ds:KeyInfo>
          <ds:KeyValue>...</ds:KeyValue>
        </ds:KeyInfo>
      </saml2:SubjectConfirmationData>
    </saml2:SubjectConfirmation>
  </saml2:Subject>
        ....
  <saml2:Statement>...</saml2:Statement>
  <saml2:Statement>...</saml2:Statement>
  ...
</saml2:Assertion>
```

Fig. 16 Sample message with sender-vouches and holder-of-key subject conformations

from unauthorized modification as well. *Holder-of-Key* enables the attesting entity to demonstrate that it is authorized to act as the subject. In other words, it confirms that the holder of the key has the same privileges as that of the subject itself. *Bearer* enables an SAML assertion to be automatically trusted by the endpoint. A consumer does not have to prove that it owns the SAML assertion. It is the simplest way to request a SAML assertion, but many endpoints do not support bearer confirmation method.

Figure 16 [23] shows a sample message with "sender-vouches" and "holder-of-key" confirmation methods using SAML security profile while Fig. 17 [23] depicts the bearer conformation method.

Sender-Vouches and Holder-of-Key conformation methods of SAML provide protection against message alteration including message insertion, deletion, and modification. However, they do not guard against eavesdropping unless coupled with some kind of an encryption mechanism such as SSL/TLS. These methods are vulnerable to replay attacks and need additional information such as times stamps, nonce and/or recipient identifiers to guard against resubmission of messages. Sender-Vouches confirmation method cannot protect a message from Man-in-the-Middle attack.

REL Token

Rights Expression Language (REL) [37] defines the rights, usage permissions, constraints, legal obligations, and license terms pertaining to an electronic document. The vocabulary for REL is defined using Open Digital Rights Language (ODRL) [38] and eXtensible Rights Markup Language (XrML) [39]. In WS-Security context [40], REL token is specified as a "license". A SOAP message contains REL licenses in its security header element.

```
<S12:Envelope xmlns:S12="...">
 <S12:Header>
  <wsse:Security xmlns:wsse="...">
   <saml:Assertion xmlns:saml="..."
     AssertionID="_a75adf55-01d7-40cc-929f-dbd8372ebdfc"
     IssueInstant="2003-04-17T00:46:02Z"
     Issuer="www.opensaml.org"
     MajorVersion="1"
     MinorVersion="1">
    <saml:AuthenticationStatement>
     <saml:Subject>
      <saml:NameIdentifier
       NameQualifier="www.example.com"
       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
       uid=joe,ou=people,ou=saml-demo,o=baltimore.com
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
       <saml:ConfirmationMethod>
        urn:oasis:names:tc:SAML:1.0:cm:bearer
       </saml:ConfirmationMethod>
      </saml:SubjectConfirmation>
     </saml:Subject>
    </saml:AuthenticationStatement>

   </saml:Assertion>

 </wsse:Security>
 </S12:Header>
 <S12:Body>
  ...
 </S12:Body>
</S12:Envelope>
```

Fig. 17 Sample message with bearer subject confirmation

Figure 18 [40] illustrates the WS-Security specification to include an REL license (lines enclosed in *<r:license>* tags) within the SOAP message header.

REL tokens can be referenced [40] using a URI either locally within a SOAP message or remotely through a URL. They can be referred to using the license identifier as well (Fig. 19).

Figure 20 [40] illustrates the signing or message parts included in the <ds:SignedInfo> element using REL license specified in licenseId of the <ds:KeyInfo> element.

KeyHolder principal of the REL security profile can provide the means to authenticate the sender of a message in which the REL license with the *KeyHolder* element is specified. The message sender can add a signature that can be verified using the

```
<S:Envelope xmlns:S="...">
  <S:Header>
    <wsse:Security xmlns:wsse="...">
      <r:license xmlns:r="...">
        ...
      </r:license>
      ...
    </wsse:Security>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Fig. 18 A SOAP message with REL license

By Location	Local	`<wsse:SecurityTokenReference>`
		`  <wsse:Reference`
		`  URI="#SecurityToken-ef375268"`
		`  />`
		`</wsse:SecurityTokenReference>`

	Remote	`<wsse:SecurityTokenReference>`
		`  <wsse:Reference`
		`  URI="http://www.foo.com/ef375268.xml"`
		`  />`
		`</wsse:SecurityTokenReference>`

By licenseId		`<wsse:SecurityTokenReference>`
		`  <wsse:Reference`
		`  URI="urn:foo:SecurityToken:ef375268"`
		`  ValueType="http://docs.oasis-open.org/wss/oasis-wss-rel-token-profile-1.0.pdf#license"`
		`  />`
		`</wsse:SecurityTokenReference>`

Fig. 19 Different methods of referencing REL licenses

key mentioned in the *KeyHolder* element providing the sender authentication. For instance, Fig. 21 demonstrates the usage of *KeyHolder* principal of a license security token to establish that the consumer whose name is mentioned in the license token is requesting a specific resource given in the message body.

Similarly, license security token can be used as an encryption key to provide confidentiality. To provide confidentiality to a message, a sender can include one or more *Encrypted Data* elements that can be decrypted using the key determined from information specified in the *KeyHolder* principal of the license token. A sender can also add an encrypted key that can be decrypted using the *KeyHolder* principal mentioned in license token. For example, Fig. 22 demonstrates the usage of an REL *KeyHolder* principal to protect the confidentiality of an XML message using the encryption key given in the *Encrypted Key* element of the security header. In this example, the original message is encrypted using the symmetric key specified in the

```
<S:Envelope xmlns:S="..." xmlns:ds="...">
 <S:Header>
  <wsse:Security xmlns:wsse="...">
   <r:license xmlns:r="..." licenseId="urn:foo:SecurityToken:ef375268"
                            xmlns:wsu="..."
                            wsu:Id="SecurityToken-ef375268">
    ...
   </r:license>
   ...
   <ds:Signature>
    <ds:SignedInfo>
     ...
    </ds:SignedInfo>
    <ds:SignatureValue>...</ds:SignatureValue>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI="#SecurityToken-ef375268"/>
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
 </S:Header>
 <S:Body>
  ...
 </S:Body>
</S:Envelope>
```

Fig. 20 Specification of WS-Security token using REL license ID

Encrypted Key element. The symmetric key given in the *Encrypted Key* element is encrypted using the recipient's RSA public key mentioned in the *KeyHolder* principal. The consumer or the recipient uses its private key to decrypt the symmetric key used for the original message encryption. After obtaining the symmetric key, the consumer uses it to decrypt the encrypted data (specified in *Encrypted Data* element) to get the actual message.

Though the use of licenses enclosed in REL tokens does not introduce any new threats, it is still vulnerable to message alteration, eavesdropping and message replay attacks. The first two problems can be resolved using an appropriate message integrity mechanism such as message signing and a confidentiality mechanism such as message encryption. Replay attacks can be addressed through the use of timestamps, caching and suitable application-specific message tracking mechanisms. REL licenses can be trusted only if they are signed natively using the mechanisms outlined in WS-Security ensuring their integrity.

WS-Addressing

WS-Addressing is an XML specification that offers capabilities to address Webservices and messages using transport-agnostic techniques. This is a WS standard specification which is used to identify webservice endpoints using XML elements. It also provides features to specify endpoint identification in messages sent across a

```
<S:Envelope xmlns:S="...">
 <S:Header>
  <wsse:Security xmlns:wsse="...">
   <r:license xmlns:r="..." licenseId="urn:foo:SecurityToken:ef375268">
    <r:grant>
     <r:keyHolder>
      <r:info>
       <ds:KeyValue>...</ds:KeyValue>
      </r:info>
     </r:keyHolder>
     <r:possessProperty/>
     <sx:commonName xmlns:sx="...">John Doe</sx:commonName>
    </r:grant>
    <r:issuer>
     <ds:Signature>...</ds:Signature>
    </r:issuer>
   </r:license>
   <ds:Signature>
    <ds:SignedInfo>
     ...
     <ds:Reference URI="#MsgBody">
      <ds:DigestMethod
       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>...</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>...</ds:SignatureValue>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference
       URI="urn:foo:SecurityToken:ef375268"
       ValueType="http://docs.oasis-open.org/wss/oasis-wss-rel-token-profile-1.0.pdf#license"/>
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
 </S:Header>
 <S:Body wsu:Id="MsgBody" xmlns:wsu="...">
  <ReportRequest>
   <TickerSymbol>FOO</TickerSymbol>
  </ReportRequest>
 </S:Body>
</S:Envelope>
```

Fig. 21 A sample SOAP message with an REL license to authenticate a consumer identified with the name "John Doe"

network of webservices in a manner which is completely independent of the protocol used at the transport layer. This feature is extremely useful when a message passes across multiple webservices through different transport protocols.

In order to provide transport neutrality, WS-Addressing uses two interoperable concepts which carry information related to transport protocols and messaging systems. These concepts translate the protocol and messaging information into a common format so that it can be processed without depending on the transport protocol

```
<S:Envelope xmlns:S="..." xmlns:ds="...">
 <S:Header>
  <wsse:Security xmlns:wsse="...">
    <r:license xmlns:r="..." licenseId="urn:foo:SecurityToken:ef375268">
      <r:grant>
       <r:keyHolder>
          <r:info>
           <ds:KeyValue>...</ds:KeyValue>
          </r:info>
       </r:keyHolder>
       <r:possessProperty/>
       <sx:commonName xmlns:sx="...">SOME COMPANY</sx:commonName>
      </r:grant>
      <r:issuer>
          <ds:Signature>...</ds:Signature>
      </r:issuer>
    </r:license>
    <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
     <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
     <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference>
          <wsse:Reference URI="urn:foo:SecurityToken:ef375268"/>
      </wsse:SecurityTokenReference>
     </KeyInfo>
     <xenc:CipherData>
      <xenc:CipherValue>dNYS...fQ=</xenc:CipherValue>
     </xenc:CipherData>
     <xenc:ReferenceList>
      <xenc:DataReference URI="#enc"/>
     </xenc:ReferenceList>
    </xenc:EncryptedKey>
  </wsse:Security>
 </S:Header>
 <S:Body wsu:Id="body"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
   <xenc:EncryptedData Id="enc"
                Type="http://www.w3.org/2001/04/xmlenc#Content"
                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
     <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
     <xenc:CipherData>
       <xenc:CipherValue>d2s...GQ=</xenc:CipherValue>
     </xenc:CipherData>
   </xenc:EncryptedData>
  </S:Body>
</S:Envelope>
```

Fig. 22 A sample SOAP message encrypted using an REL license

or application. The two constructs are *endpoint references* and *message information headers*. *Endpoint references* (EPR) present the information needed to identify and invoke the operations provided by a webservice provider. *Message information headers* are used to address messages uniformly irrespective of underlying transport and to carry the end-to-end characteristics of a message that includes the message

```
(001) <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
          xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(002)   <S:Header>
(003)    <wsa:MessageID>
(004)      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
(005)    </wsa:MessageID>
(006)    <wsa:ReplyTo>
(007)      <wsa:Address>http://business123.example/client1</wsa:Address>
(008)    </wsa:ReplyTo>
(009)    <wsa:To>http://fabrikam456.example/Purchasing</wsa:To>
(010)    <wsa:Action>http://fabrikam456.example/SubmitPO</wsa:Action>
(011)   </S:Header>
(012)   <S:Body>
(013)     ...
(014)   </S:Body>
(015) </S:Envelope>
```

Fig. 23 A SOAP message with endpoint references and message information headers

identifier along with the addresses of source and destination endpoints. These two concepts together provide the capability to communicate addressing information needed for delivering a message to different webservices. The basic components of a message address are—a *source endpoint* from which the message has originated, a *destination endpoint* to which the message should be delivered, a *reply endpoint* to which any replies should get dispatched, a *fault endpoint* to which any exception messages are sent, an *action* that describes the operation that needs to be performed on the message, a *message ID* to identify the message uniquely and any *relationship information* with previous messages. WS-Addressing feature addresses the message replay attacks by specifying a unique ID to the message that cannot be duplicated. It can also specify the source and destination addresses for the message to prevent the replay attacks from a different source.

Figure 23 [41] illustrates the use of these mechanisms in a SOAP 1.2 message being sent from http://business123.example/client1 to http://fabrikam456.example/Purchasing to call the action "SubmitPO".

As illustrated in Fig. 23, the SOAP message header is specified with the WS-Addressing information. The line 004 represents the unique message ID while line 007 specifies the reply endpoint to which any reply to this message should be dispatched and line 009 provides the destination endpoint where this message should be delivered to. Line 010 describes the operation that should be executed when this message is sent to the destination endpoint. The actual message body is represented in lines (012) to (014).

Anatomy of WS-Security-Enabled SOAP Message

As discussed in the previous sections, WS-Security addresses the security of webservices by adopting the existing standards and standards-based technologies in transport-protocol agnostic way. WS-Security uses XML signature to provide

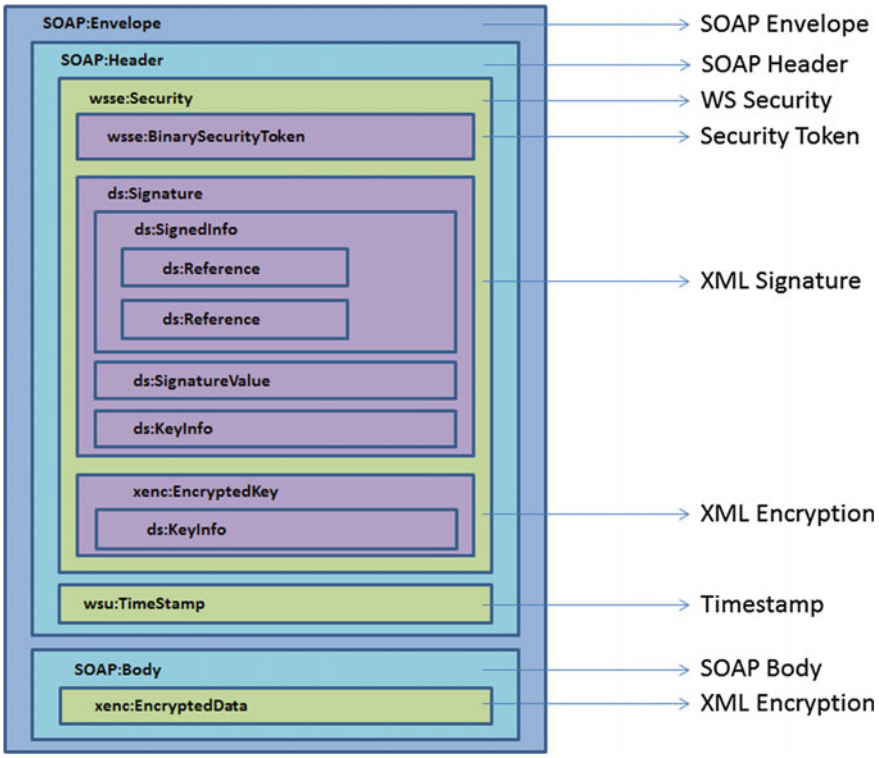


Fig. 24 Anatomy of a SOAP message

integrity and XML encryption to provide confidentiality while it makes use of various security tokens such as username tokens, binary security tokens and XML-based tokens to provide authentication and authorization for a SOAP message. Based on these security profiles, it is worthwhile to have a closer look at the SOAP messaging structure enriched with WS-Security along with its core elements.

WS-Security attaches a set of security extensions within a SOAP message describing the structure, its core elements, processing instruments and rules to enable the message-level security. WS-Security-specific elements are specified as the child elements of SOAP header along with their namespaces. Figure 24 gives a succinct picture of a SOAP message enriched with various elements of WS-Security framework [42].

Figure 25 [43] gives a comparison of various WS-Security mechanisms and the threats addressed by them.

Countermeasure \ Security Threat	Message Alteration	Loss of Confidentiality	Falsified Message	Man in the Middle	Principal Spoofing	Forged Claims	Replay of Message Part	Replay of Message	Denial of Service
XML Encryption		X		X	X	X	X		
XML Signature	X		X		X	X	X	X	
WS-Security Tokens			X		X	X			
WS-Addressing								X	
SSL/ TLS	X	X	X*	X	X*	X*	X		
SSL/ TLS with Client Certificates	X	X	X	X	X	X	X		
HTTP Authentication			X		X	X			

X → Security threat is fully addressed by the given countermeasure
X* → Security threat is partially addressed by the given countermeasure

Fig. 25 Security mechanisms and threats addressed by them

4 Other Security Frameworks

Apart from the security measures given in the aforementioned sections, there are other standards and specifications aimed at supporting the protective measures hitherto discussed. Among them, XML Key Management System (XKMS) and Extensible Access Control Markup Language (XACML) are the most prominent standards. XML Key Management System (XKMS) [44, 45] is a standard which defines the protocols to take care of the public key infrastructure. It is an XML-based standard to register and distribute public keys used in digital signatures and encryption. XKMS [46] consists of two child specifications—XML Key Registration Service Specification (X-KRSS) for registering public keys and XML Key Information Service Specification (X-KISS) for validating keys provided in XML signature. Extensible Access Control Markup Language (XACML) [47, 48] is another specification which aims at enhancing the access control capability of Webservices. XACML uses access control matrix model which defines authorization rules for accessing an XML document either in parts or as a whole.

XKMS

XKMS is a World Wide Web Consortium (W3C) standard, which outlines the methodology to access and integrate Public Key Infrastructure (PKI) [49]. PKI lays the security foundation for creating XML signatures and exchanging XML messages in an encrypted format. Different PKI solutions include X.509, Simple Public Key Infrastructure (SPKI) [50], Pretty Good Privacy (PGP) [50], and Public Key Infrastructure (PKIX) [51]. Though PKIs provide the most robust security framework,

choosing the right one is a challenge. Managing PKIs becomes more complex when the sender and the receiver use different key specifications. For instance, when a sender sends a message in X.509 encryption format to a receiver that is using SPKI, the receiver will not be able to decrypt the message. So, in order to exchange messages, both the sender and the receiver have to understand each other's PKI solution. Extending the scenario to multiple webservices, we need to ensure that all the webservices that want to collaborate have to be aware of one another's PKI solution. XKMS provides a solution to resolve this issue.

XKMS provides the framework to manage the PKI through a trusted third party and decouples the PKI management from client applications including webservices. The third party provides a PKI interface to different client applications including webservices by hosting the XKMS service. The basic goals of XKMS are i) to create an abstraction layer between an application or a webservice and the PKI so that applications can be integrated with different PKI solutions without necessitating any changes, ii) to remove the unnecessary overhead of understanding complex PKI syntax and semantics from applications thereby allowing them to focus only on application functionality and iii) to use platform-independent, transport-neutral, and vendor-agnostic techniques to integrate various PKI solutions with applications.

As described before, XKMS is made up of two subprotocols—XKRSS and XKISS. The former is key registration service while the latter is key validation service.

XML Key Registration Service Specification (XKRSS)
The first portion of XKMS is used to register a key pair with a client application. There are two ways to register a key pair with an XKMS provider. One way is to let the client application generate the key pair and register its public key with an XKMS provider. The second way is to let the XKMS provider generate the key pair on behalf of a client application and register the public key with itself while sending the private key to the client application. In both the cases, XKMS service provider may keep the private key with itself based on the consent from the client application so that it can be recovered if it is lost by the client application.

This specification encompasses four operations: "Register"—that allows the clients to register their public keys and optionally the private keys, "Reissue/Renew"—that allows an XKMS provider to reissue/renew the previously issued key pair by generating new credentials for the key pair, "Revoke"—that permits clients to delete the data objects pertaining to a key pair and "Recover"—that allows clients to recover their private keys if the private keys are registered with the XKMS service provider.

XML Key Information Service Specification (XKISS)
The second part of XKMS is used to allow client applications to validate encrypted or signed messages. A client (which is a web application or a webservice) authenticates an encrypted/signed message by sending the key details embedded in the message to an XKMS service provider. The XKMS service provider then validates the key and ratifies whether the key pair used for encryption or signing is valid and indeed belongs to the entity that has done the encryption or signing.

```
<Register>
  <Prototype Id="keybinding">
    <Status>Valid</Status>
    <KeyID>mailto:Alice@cryptographer.test</KeyID>
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
          998/T2PUN8HQInhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
          fdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/I6xw
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
      <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
    </ds:KeyInfo>
    <PassPhrase>Pass</PassPhrase>
  </Prototype>
  <AuthInfo>
    <AuthUserInfo>
      <ProofOfPossession>
        <ds:Signature URI="#keybinding"
            [RSA-Sign (KeyBinding, Private)] />
      </ProofOfPossession>
      <KeyBindingAuth>
        <ds:Signature  URI="#keybinding"
            [HMAC-SHA1 (KeyBinding, Auth)] />
      </KeyBindingAuth>
    </AuthUserInfo>
  </AuthInfo>
  <Respond>
    <string>KeyName<string>
    <string>KeyValue</string>
    <string>RetrievalMethod</string>
  </Respond>
</Register>
```

Fig. 26 Sample XKMS key registration request message

An XKISS specification has two operations: “Locate”—that resolves the key used for encryption or signing but does not prove the validity of the key and “Validate”—that not only resolves the key used for encryption or signing but also returns whether it is valid or not.

Let us look at some of the sample messages sourced from [44] to illustrate the use of XKMS. Figure 26 shows a sample request to register a client-generated key with an XKMS server.

In Fig. 26, *Prototype* element encapsulates the key information associated with a client identified by the email ID “*Alice@cryptographer.test*”. The corresponding

```
<RegisterResult>
  <Result>Success</Result>
  <Answer>
    <Status>Valid</Status>
    <KeyID>mailto:Alice@cryptographer.test</KeyID>
    <ds:KeyInfo>
      <ds:RetrievalMethod
         URI="http://www.PKeyDir.test/Certificates/01293122"
         Type="http://www.w3.org/2000/09/xmldsig#X509Data"/>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Ef
          dxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
      <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
    </ds:KeyInfo>
  </Answer>
</RegisterResult>
```

Fig. 27 Sample XKMS key registration response message

public key is identified by a combination of modulus and exponent. *Passphrase* element authenticates the user who submitted the request, against the XKMS key registration service. *Proof Of Possession* element guarantees that private key corresponding to the given public key is possessed by the client. This element consists of the digital signature of the element *Prototype* signed using the private key possessed by the client. If the digital signature of *Prototype* enclosed under *Signature* element can be validated using the public key given in *KeyInfo*, it must have been signed using the corresponding private key. *KeyBidnignAuth* element authenticates the given client request. This done by validating the digital signature of *Prototype* using an authentication code shared between the client and the XKMS service provider for this particular request. The *Respond* element indicates that the client is expecting the key name and the public key sent in the request along with a method to retrieve the public key in future.

Figure 27 illustrates a sample response sent by an XKMS provider for the request shown in Fig. 26. As requested, the XKMS provider responds with the given key name and the public key along with a retrieval method. In this case, the public key can be retrieved using an X.509 certificate located in the given URI.

Figure 28 illustrates a sample message to retrieve the key name and the public key from a given X.509 certificate. Figure 29 shows the key name and the public key retrieved for this request.

Locate method does not check the validity of a given key. It just requests the XKMS provider to return the key details for a given request. In order to check the validity of a key, we need to send a validate request. Figure 30 shows a validate request message while Fig. 31 shows the corresponding response message. <Result> element indicates that the request is successfully processed and the validity interval

```
<Locate>
  <Query>
    <ds:KeyInfo>
      <ds:RetrievalMethod
        URI="http://www.PKeyDir.test/Certificates/01293122"
        Type="http://www.w3.org/2000/09/xmldsig#X509Data"/>
    </ds:KeyInfo>
  </Query>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
  </Respond>
</Locate>
```

Fig. 28 Sample key locate request

```
<LocateResult>
  <Result>Success</Result>
  <Answer>
    <ds:KeyInfo>
      <ds:KeyName>O=XMLTrustCernter.org OU="Crypto"
            CN="Alice"</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Ef
          dxSXAidruAszNqBoOqfarJlsfcVKLob1hGnQ/l6xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </Answer>
</LocateResult>
```

Fig. 29 Sample locate response message

enclosed under <ValidityInterval> tag confirms that the given key is valid for one month from 20th Sep 2000 to 20 Oct 2000.

XACML

eXtensible Access Control Markup Language (XACML) [47, 48, 52] is an XML-based declarative language to process access control requests. It defines methods to evaluate access requests according to the rules specified in policies. XACML supports both Attribute-Based Access Control (ABAC) mechanism and Role-Based Access Control (RBAC) mechanism. ABAC is a system wherein attributes associated with a user/action/resource combination determine whether a given user has access to perform a given action on a given resource. RBAC is a specialization of ABAC and checks if a group of users tagged under a role can perform a given action on a given resource according to the policies defined for the given role.

XACML is built on the principle of segregation of duties between the action decision and the point of action. Action decision is the mechanism that decides if a given action is permitted for a given user on a given resource according to the

```
<Validate>
  <Query>
    <Status>Valid</Status>
    <ds:KeyInfo>
      <ds:KeyName>O=XMLTrustCernter.org OU="Crypto" CN="Alice"</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Ef
          dxSXAidruAszNqBoOqfarJlsfcVKLob1hGnQ/l6xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ ds:KeyInfo>
  </Query>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
  </Respond>
</Validate>
```

Fig. 30 Sample validate request message

```
<ValidateResult>
  <Result>Success</Result>
  <Answer>
    <KeyBinding>
      <Status>Valid</Status>
      <KeyID>http://www.xmltrustcenter.org/assert/20010120-39</KeyID>
      <ds:KeyInfo>
        <ds:KeyName>O=XMLTrustCernter.org OU="Crypto" CN="Alice"</ds:KeyName>
        <ds:KeyValue>
          <ds:RSAKeyValue>
            <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSX
AidruAszNqBoOqfarJlsfcVKLob1hGnQ/l6xw</ds:Modulus>
            <ds:Exponent>AQAB</ds:Exponent>
          </ds:RSAKeyValue>
        </ds:KeyValue>
      </ ds:KeyInfo>
      <ValidityInterval>
        <NotBefore>2000-09-20T12:00:00</NotBefore>
        <NotAfter>2000-10-20T12:00:00</NotAfter>
      </ValidityInterval>
    </KeyBinding>
  </Answer>
</ValidateResult>
```

Fig. 31 Sample validate response message

rules defined in policies. Point of action is the execution of the action itself. On the contrary, if the decision-making system is embedded in a client application, it becomes difficult to update the decision criteria when the governing policy changes. If the access decisions are decoupled from a client application, decision policies can be updated on the fly and affect all the clients that use the policy.

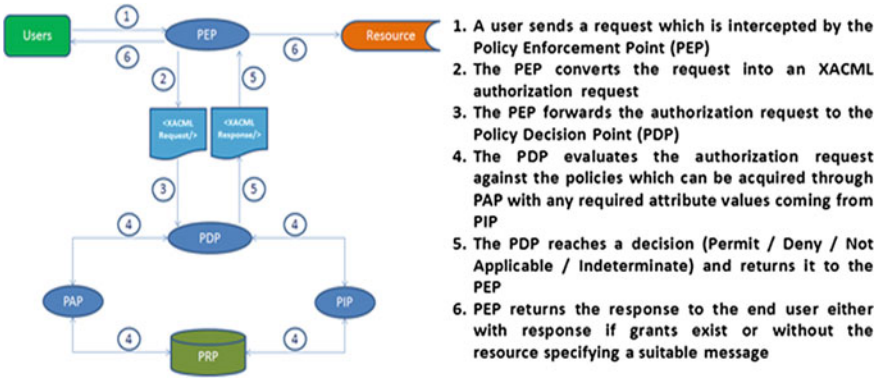


Fig. 32 XACML validation process

XACML ecosystem is built using a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), a Policy Administration Point (PAP), a Policy Information Point (PIP), and a Policy Retrieval Point (PRP). When a consumer wants to access a resource, the XACML system validates and grants the required accesses as illustrated in Fig. 32.

XACML consists of PolicySet, Policy and Rule in a hierarchy to work on Attribute Categories. A policy set is a collection of policy elements and a policy is a collection of many rule elements.

Attribute Categories:
XACML elements work on attribute categories called subjects, resources, and actions. A subject is an entity that requests access, a resource is a data, service, or system component and an action is the type of access requested by a subject on a resource. All these elements are defined as one or more attributes.

Rule:
A rule is the most basic element of a policy and consists of a *target*, a *condition* (optionally), an *effect*, an *obligation* (optionally) and/or an *advice* (optionally). A *target* is a set of requests to which the rule is intended to apply and is specified in the form of a logical expression on attributes in the request. A *condition* is a Boolean expression that determines if the given rule is applicable or not. An *effect* of a rule determines the consequence of an evaluation and it either permits or denies the action associated with a given rule. If the condition associated with a rule evaluates to "True", the effect of the rule is executed. If the effect is "Permit", the corresponding action is permitted and if the effect is "Deny", the corresponding action is denied. If the evaluation of a condition results in an error, the rule goes into "Indeterminate" status and returns a processing error. An *obligation* is a directive that enforces PEP to perform the given action mandatorily before or after the access is approved. An *advice* is similar to an obligation except that it does not enforce PEP to perform the action specified in the advice.

Policy:
A policy is essentially a collection of rules and consists of a target, a rule-combining algorithm, and an optional list of obligation and/or advice expressions. Policy targets are similar to rule targets except that a policy target applies to all the rules enclosed in the policy by default. A rule-combining algorithm specifies the procedure by which the outcomes of rule evaluations are combined. Policy evaluation is guided by the sequence of rule combinations and their evaluations. This process is executed by PDP according to the given rule-combining algorithm. Obligation and advice expressions are similar to those belonging to rules except that they apply to all the rules of the policy.

Policy Set:
A policy set encapsulates many policy elements and can contain a target, a policy-combining algorithm and a list of obligatory and advisory expressions optionally. Targets, obligation expressions, and advice expressions are similar to those defined in rules and policies except that they apply to the entire policy set by default. The policy-combining algorithm specifies the mechanism to combine the decisions and obligations from multiple policies embedded in the given policy set. Based on the given policy-combining algorithm, the outcome of the policy set is evaluated and placed in response by PDP.

Figure 33 shows a sample policy [47] that applies to requests for the server called "SampleServer". A specific rule of this policy has a target that requires an action called "login" and a condition that applies only if a subject is trying to login between "9:00 AM" and "5:00 PM". It also has a default rule which denies all other requests.

5 An Overview of Oracle Web Services Management (OWSM)

Oracle Web Services Manager (OWSM) [53] is a standards-based complete security solution for providing security to webservices orchestrated in Service-Oriented Architecture (SOA). Oracle WSM has become a primary choice for organizations because it allows (1) configuring attachable security policies for securing webservices in a centralized manner, (2) imposing security management policies through customizable agents and (3) monitoring security incidents in real time. These key features provide organizations with the necessary agility to respond to any security incidents or breaches and to make necessary changes in security policies online without interrupting any of its ongoing business processes.

OWSM Architecture
Oracle Web Services Manager (OWSM) is a policy-based policy framework to manage the security of webservices consistently in service-oriented architecture. This framework is based on WS-Policy standard and uses a declarative language. It offers the necessary capabilities to develop, impose, execute, and monitor webservice security policies that include authentication, authorization, reliable messaging,

```
<Policy PolicyId="SamplePolicy"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

  <!-- This Policy only applies to requests on the SampleServer -->
  <Target>
   <Subjects>
    <AnySubject/>
   </Subjects>
   <Resources>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
     <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
    </ResourceMatch>
   </Resources>
   <Actions>
    <AnyAction/>
   </Actions>
  </Target>

  <!-- Rule to see if we should allow the Subject to login -->
  <Rule RuleId="LoginRule" Effect="Permit">
   <!-- Only use this Rule if the action is login -->
   <Target>
    <Subjects>
     <AnySubject/>
    </Subjects>
    <Resources>
     <AnyResource/>
    </Resources>
    <Actions>
     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
      <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                    AttributeId="ServerAction"/>
     </ActionMatch>
    </Actions>
   </Target>
   <!-- Only allow logins from 9am to 5pm -->
   <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
     </Apply>
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
     </Apply>
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
    </Apply>
   </Condition>
  </Rule>
  <!-- We could include other Rules for different actions here -->
  <!-- A final, "fall-through" Rule that always Denies -->
  <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Fig. 33 A sample XACML policy

message transmission optimization mechanism (MTOM), and addressing. OWSM policy framework consists of a *Policy Manager*, an *Agent*, an *OWSM Repository,* and an *Enterprise Manager*. *Policy Manager* can create and update both predefined and custom policies from OWSM repository. *Agent* is made up of several library (JAR) files and is loaded on to the server on which OWSM is deployed. It is a state-less entity that enforces and executes policies using an in-memory policy cache. Agent consists of Policy Access Point (PAP) and Policy Interceptor. Policy Access Point communicates to the Policy Manager and loads the policies in memory. Policy Interceptor gets created when a new webservice is deployed and policy is attached to a webservice and is responsible to enforce a policy. *OWSM Repository* stores all the predefined and custom policies, and can be accessed and updated through Policy Manager. *Enterprise Manager* provides the capability to configure OWSM.

OWSM architecture is depicted in Fig. 34. According to this architecture, when a webservice secured by an OWSM policy is accessed by a consumer, the given request is initially delivered to the OWSM agent. This agent extracts the security information from the request and queries the OWSM policy cache to impose the applicable policies on the request. If the cache does not have policy information related to the security information provided in the request, the OWSM agent connects to the OWSM Policy Manager to fetch the related policies into the cache. After fetching the applicable policies, the agent enforces these policies on the request. These policies may authenticate, encrypt, decrypt, authorize or log the given request according to their applicability. The OWSM agent, by default, only works with cached policies and does not connect to the Policy Manager. However, if there is a configuration change in any of the policies, the OWSM Agent connects to Policy Manager to reload the updated policies into the cache.

Policies and Assertions

A policy is a condition under which an operation of a webservice can be granted to a consumer. The WS-Policy framework [54] specifies policy information that can be managed by webservice security applications like OWSM. Policy expressions are defined using WS-Policy Attachment specification [55] on various webservice elements specifying security concerns for these elements. These policies are embedded in a WSDL file and stored in UDDI for SOA so that external agencies like service consumers can be integrated into webservices securely. A policy is a collection of one or more policy assertions. A policy assertion is the basic unit of a policy and performs an action applicable to the given request and the corresponding response. For instance, a policy assertion may instruct that any given request to a webservice has to be encrypted. Likewise, another policy assertion may mandate that both request and response messages have to be logged. Assertions of a policy are chained together in a pipeline. When a webservice is attached to a policy, the assertions of the policy are executed both on a given request and its response. When a request is made to a service, the assertions given in the policy attached are executed one after the other in the same sequence in which they appear in the policy. If the service gives a response for the given request, all the assertions of the same policy are executed in the reverse order on the response. Figure 35 illustrates that *assertion 1* is executed followed by

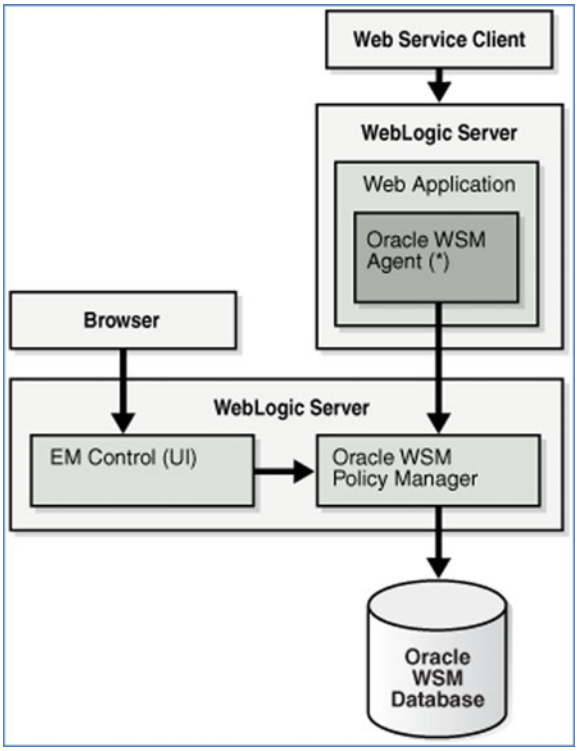


Fig. 34 OWSM architecture

assertion 2, and so on up to *assertion n* for a request message whereas the same set of assertions are executed in the reverse order on the response message if one exists. Figure 36 shows a sample OWSM policy with two assertions. The *wss11-username-with-certificates* assertion authenticates a consumer based on username and password given in the WS-Security Username Token element of the request while the *binding-authorization* assertion provides access to the authenticated consumer to execute a specific operation granting the resources needed. If there is a response from the webservice, the *binding-authorization* assertion is applied first followed by *wss11-username-with-certificates* assertion though these assertions do not impose any restrictions on the response. Figure 37 illustrates the steps involved in a policy execution.

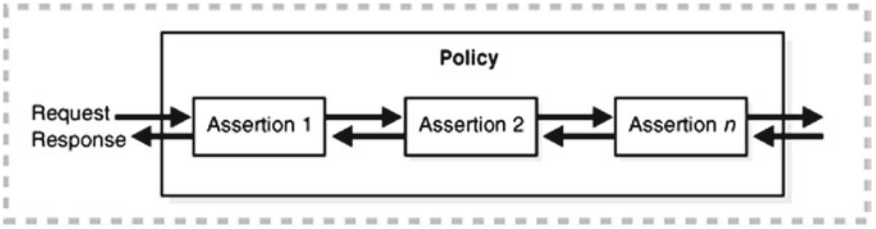


Fig. 35 OWSM policy with multiple assertions

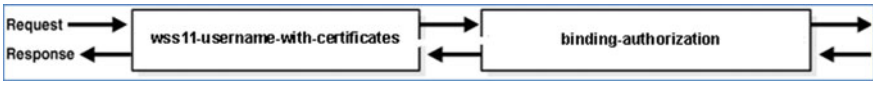


Fig. 36 A sample OWSM policy with multiple assertions

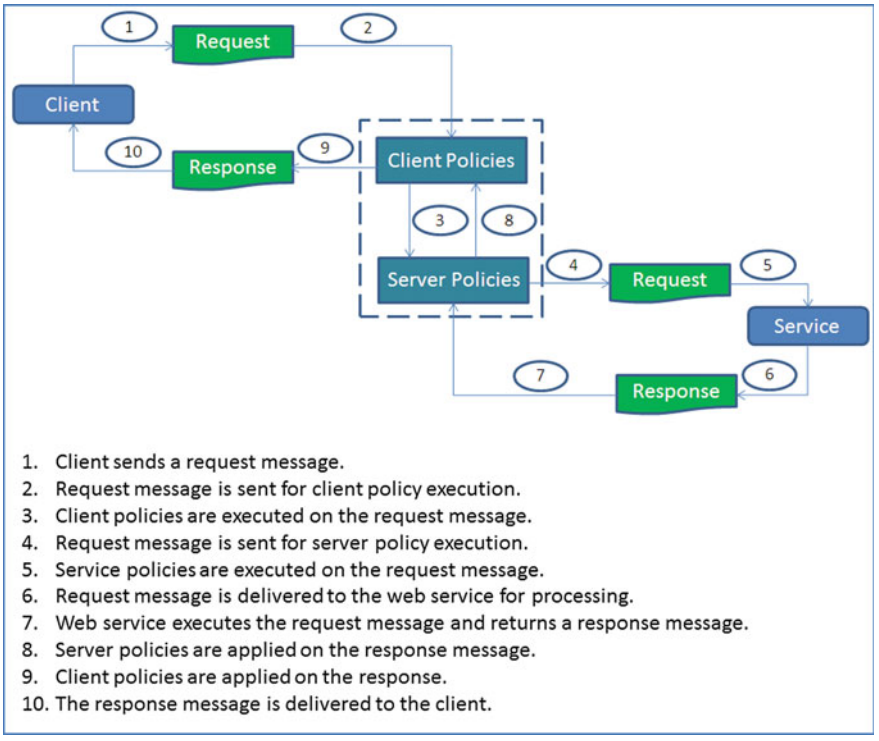


Fig. 37 OWSM policy execution

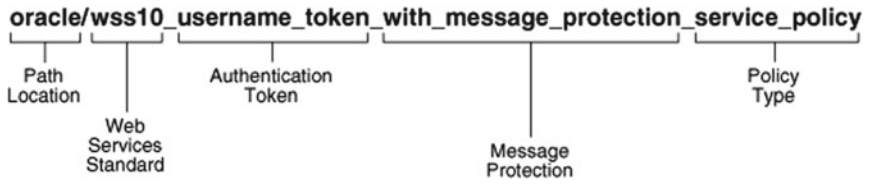


Fig. 38 OWSM policy naming convention

OWSM Policy Categories
Policies are broadly classified into the following categories:

1. *Security*: These policies are based on WS-Security 1.0 and 1.1 standards. Both these policies mainly aim at providing authentication and authorization for web-service providers and consumers, and protection for messages. These policies support all the security tokens discussed in the previous sections of this chapter.
2. *WS-Addressing*: These policies are built using WS-Addressing standards and use WS-Addressing headers in SOAP messages. They manage the ordered exchange of messages between a provider and a consumer to thwart message replay attacks.
3. *Reliable Messaging*: These policies are built using WS-Reliable Messaging [56] standard and aim at guaranteed and ordered delivery of messages between a provider and a consumer. These policies are useful in scenarios where messages need to be delivered in a specific order. They can instruct the receiving system to process the incoming messages in the correct order even though they arrive in a different order. They can also be configured to ensure that messages are delivered exactly once thereby avoiding replay attacks.
4. *Management*: Management policies work toward maintaining logs for request, response, and fault messages for future auditing.

Though OWSM supports a few more policy categories, they are specifically meant to support interoperability and are out of scope for this chapter. Readers are advised to go through Oracle documentation on OWSM [53] for more details on these policies.

Policy Naming Convention
Oracle recommends a naming convention to define OWSM policies so that the policy name itself illustrates its usage. Figure 38 depicts the convention to define an OWSM policy though this is not strictly enforced by Oracle:

Path Location: This specifies the directory wherein the given policy is available. For example, all predefined policies of OWSM are generally available in the directory named oracle.
Webservices Standard: This part specifies the WS-Security standard used in the given policy. It can be set as "*wss10*" if the policy is based on WS-Security 1.0 [57] or "*wss11*" for WS-Security 1.1 [58] or "*wss*" for supporting either of the versions.
Authentication token: This part indicates the type of security token used in authentication. OWSM by default supports http_token, kerberos_token, saml_token, username_token and X509_token.

Transport security: This portion specifies if the policy needs that messages are to be transported over a secure protocol. In this case, the token name is suffixed with *over_ssl*. For instance, wss_http_token_over_ssl indicates that the HTTP security token needs to be transmitted through SSL at the transport layer.
Message protection: This piece indicates the policy needs to enforce message protection by providing integrity and confidentiality. The policies that need message protection are appended with the phrase with_message_protection.
Policy Type: This part indicates if the given policy needs to be executed either at the consumer or at the provider. It also specifies whether the given policy is a policy or a template.

A Few Predefined Policies
Though there are several categories of policies defined by OWSM, let us look at only those which are most frequently used. There are two types of security policies for providing authentication. The first set of policies called service policies to provide authentication for a consumer service while invoking a provider service. These policies request for credentials in the inbound request and validate them against the security realm defined by the server in which the provider service is deployed. For instance, wss_http_token_service_policy is a service policy that adds HTTP basic authentication policy to a provider service so that any consumer that wants to invoke the provider needs to supply valid credentials in the HTTP header section of its request. This policy can be applied to both RESTful and SOAP services. Similarly, the service wss_username_token_service_policy can be used to add WS-Security information in WS header section of a SOAP request to authenticate a consumer service. Any consumer invoking the provider service added with this policy needs to supply valid credentials in plain text format in its request. There are several sets of such policies relating to authorization, message protection, message auditing, and message addressing. Readers are advised to read the corresponding documentation available at reference [53] to understand more about these predefined policies. All these policies are self-explanatory in the sense that the name of a policy itself indicates its applicability and usage as it follows the nomenclature mentioned in the previous section.

Examples
This section illustrates a couple of examples that describe the use of OWSM policies in securing webservices. The first example adds HTTP basic authentication for RESTful services. This service is built using Oracle Service-Oriented Architecture components (BPEL and Mediator) as shown in Fig. 39.

The binding component that invokes this service is GetOrdersRS. This endpoint is secured through an HTTP basic authentication policy by adding the predefined policy “oracle/wss_http_token_service_policy” which is shown in Fig. 40.

If we try to execute or invoke this service without supplying valid credentials, the invocation fails with an authorization error. The request and the response are shown in Fig. 41 and Fig. 42 respectively.

Fig. 39 A RESTFul service composite definition (GetOrders) with endpoint GetOrders RS

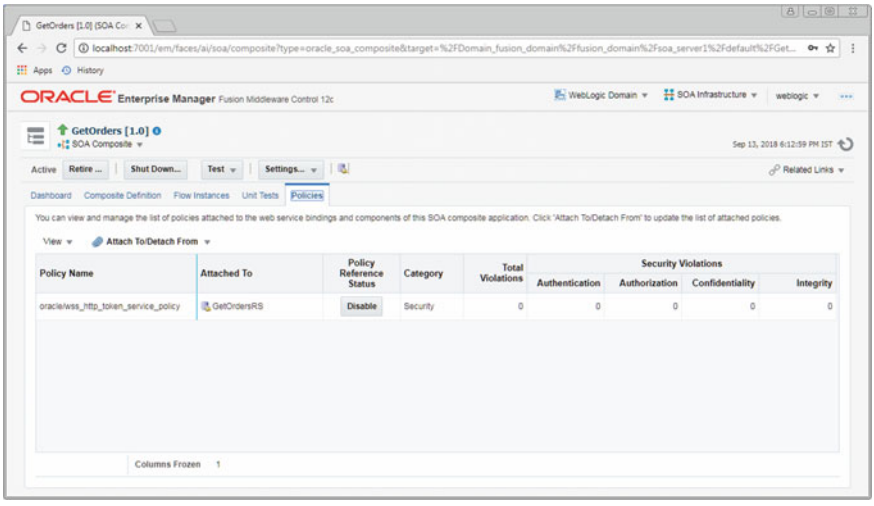


Fig. 40 Addition of HTTP basic authentication policy

The second example shows how a WSS username token can be added to a SOAP-based service. This is also a SOAP-based service composite built using BPEL component as shown in Fig. 43.

The binding component of this service composite is QueryService_ep which takes a SOAP request as the input. This endpoint is secured through a WSS username token authentication policy by adding the predefined policy "oracle/wss_username_token_service_policy" which is shown in Fig. 44.

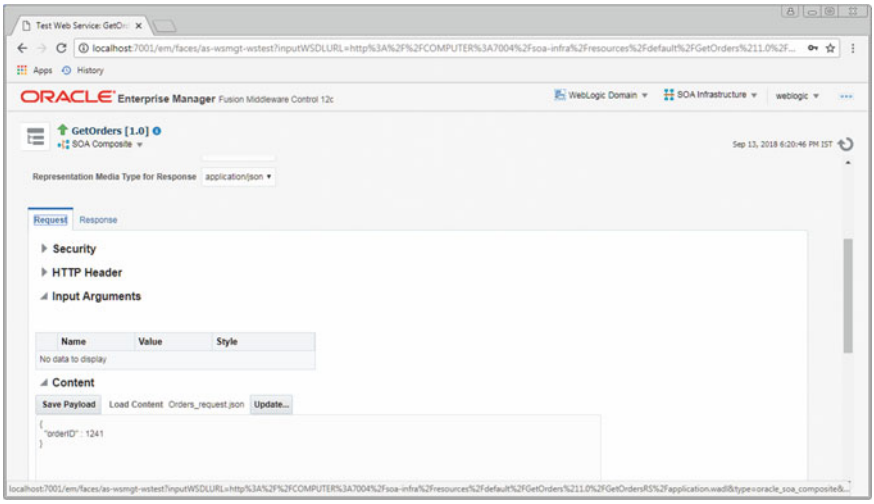


Fig. 41 Sample request to invoke the GetOrders service composite

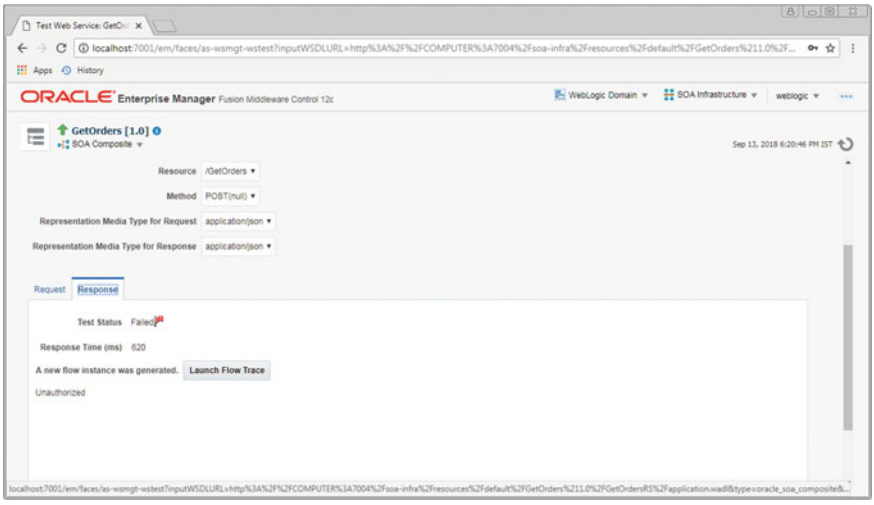


Fig. 42 Response indicating that the service is invokes using unauthorized credentials

As in the case of the first example, if we try invoking the QueryService using no credentials or invalid credentials, the invocation fails with the appropriate fault message. Figure 45 illustrates the error message we get.

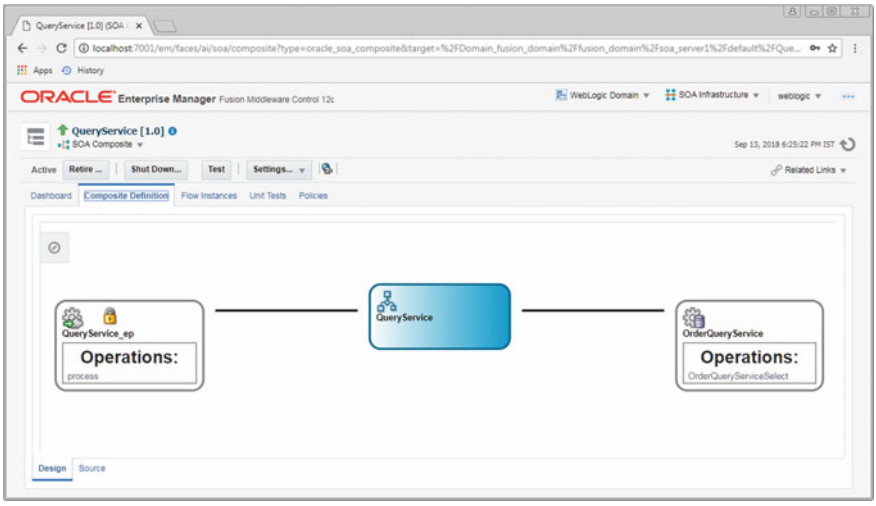


Fig. 43 A SOAP-based service composite (QueryService) with endpoint QueryService_ep

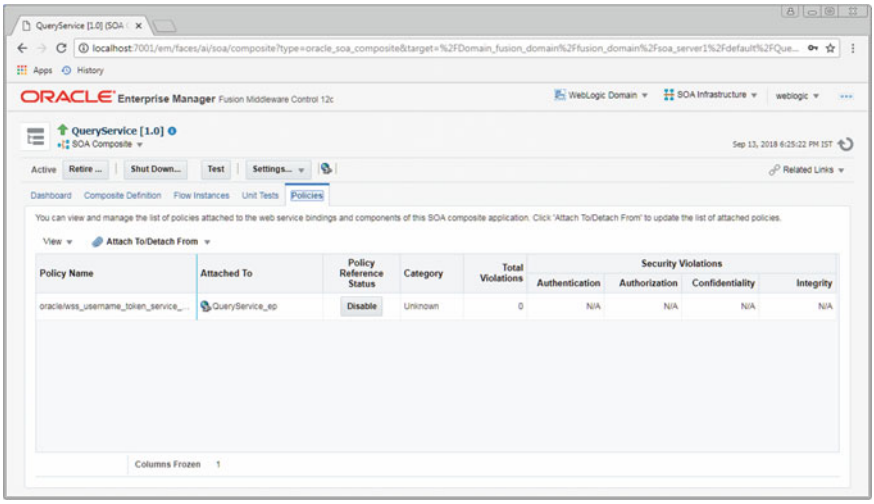


Fig. 44 Adding WSS username token policy

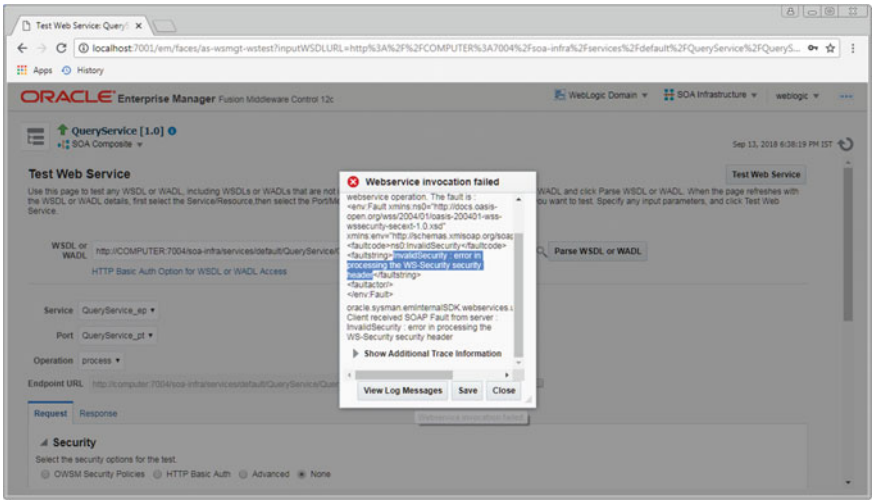


Fig. 45 Error while invoking the service QueryService with invalid credentials

6 Conclusion

This chapter begins with an introduction to webservices followed by an anatomy of a webservice using Web Service Description Language (WSDL). It then gives an overview of their classification based on the messaging protocol that they use—-SOAP which is a tightly coupled traditional framework and REST which is a loosely coupled modern framework. It then explains the synchronous request–reply, asynchronous request–reply, and one-way exchange patterns in webservice producer and consumer interactions. It then gives an introduction to security fundamentals that include Authentication, Authorization, Non-repudiation, Confidentiality, Integrity, and Availability. It then explains the security nomenclature associated with security in general. After a brief summary of various security threats associated with webservices, the chapter gives a detailed elucidation and a comparison of different security countermeasures that include XML digital signatures, XML encryption, WS-Security tokens and WS-addressing to address various threats. It basically divides the security measures into two categories—Transport Layer Security and Message-Level Security. Transport layer security uses SSL/TLS to provide confidentiality, integrity and provider authentication. It uses basic or digest authentication mechanisms over HTTP or HTTPS to provide consumer authentication. Message-level security uses XML digital signatures to ensure integrity and non-repudiation, XML encryption to provide confidentiality and WS-Security tokens to ascertain the sender's identity. This chapter gives a detailed presentation and usage of WS-Security tokens within SOAP framework. It also presents an overview of the more recent security standards such as XKMS and XACML along with a few examples. It finally concludes giving an introductory overview of the policy-based framework of OWSM from Oracle.

References

1. http://www.sciencedirect.com/science/article/pii/S0020025514005428
2. http://www.pressenet.info/texte/service-oriented-architecture.html
3. http://www.ws-i.org/profiles/basicsecurity/securitychallenges-1.0.pdf
4. https://www.w3.org/TR/soap/
5. https://www.w3.org/2002/ws/cg/2/07/meps.html
6. https://www.infosec.gov.hk/english/technical/files/webss.pdf
7. https://en.wikipedia.org/wiki/OASIS_(organization)
8. https://msdn.microsoft.com/en-us/library/ff648318.aspx
9. http://research.ijcaonline.org/icsem/number1/icsem1324.pdf
10. http://www.xyzws.com/scdjws/WSGEN/4
11. https://www.tomsguide.com/us/ssl-vs-tls,news-17508.html
12. https://www.digicert.com/blog/evolution-of-ssl/
13. https://www.digicert.com/ssl.htm
14. https://pdfs.semanticscholar.org/b111/0264f5efa9848bfa647cc8f7f8ea7ecebc34.pdf
15. https://tools.ietf.org/html/rfc3174
16. http://euler.ecs.umass.edu/ece597/pdf/Crypto-Part12-MAC.pdf
17. https://access.redhat.com/documentation/en-US/Fuse_ESB_Enterprise/7.1/html/ActiveMQ_Security_Guide/files/X509CertsWhat.html
18. http://www.webdav.org/specs/rfc2617.html
19. https://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
20. http://www.herongyang.com/Web-Services/X509-Token-WSS-X509-Certificate-Token-Profile.html
21. https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf
22. https://msdn.microsoft.com/en-us/library/bb742516.aspx
23. http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SAMLTokenProfile-v1.1.1.html
24. http://www.oracle.com/technetwork/articles/java/dig-signatures-141823.html
25. https://www.xml.com/pub/a/2001/08/08/xmldsig.html
26. https://www.cs.virginia.edu/~acw/security/doc/Tutorials/WS-Security.ppt
27. https://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html
28. https://docs.oracle.com/cd/E27515_01/common/tutorials/encryption_encrypt_settings.html
29. https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf
30. https://www.di-mgt.com.au/rsa_alg.html
31. https://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf
32. https://tools.ietf.org/html/rfc2315
33. http://www.zeroshell.org/kerberos/Kerberos-operation/
34. http://web.mit.edu/kerberos/
35. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
36. https://docs.oracle.com/cd/E24001_01/web.1111/b32511/standards.htm
37. http://xml.coverpages.org/LOC-CoyleReportREL2004.pdf
38. https://www.w3.org/TR/odrl/
39. http://www.service-architecture.com/articles/web-services/extensible_rights_markup_language_xrml.html
40. http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-rel-token-profile-v1.1.1-os.html
41. https://www.w3.org/Submission/ws-addressing/
42. http://www.cs.ucsb.edu/~bultan/courses/595-W06/WS-Security.ppt
43. https://www.owasp.org/images/3/33/Web_Services_Security_%E2%80%93_Challenges_and_Trends.ppt
44. https://www.w3.org/TR/xkms/
45. http://www.exchangenetwork.net/node/mentoring/2005_meeting/xkms.ppt
46. https://www.ibm.com/developerworks/xml/library/x-seclay3/x-seclay3-pdf.pdf

47. https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html
48. https://en.wikipedia.org/wiki/XACML
49. http://www.au-kbc.org/bpmain1/PKI/PKIieee.pdf
50. http://www.facweb.iitkgp.ernet.in/~sourav/PGP.pdf
51. https://www.limited-entropy.com/docs/spki.pdf
52. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
53. https://docs.oracle.com/middleware/1212/owsm/OWSMC.pdf
54. https://www.w3.org/Submission/WS-Policy/
55. https://www.w3.org/Submission/WS-PolicyAttachment/
56. http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html
57. https://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
58. http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-SOAPMessageSecurity-v1.1.1-os.html

Webservices Engineering

Venkata Swamy Martha and Maurin Lenglart

Abstract A webservice is a loosely coupled business process accessible over web and often a webservice leverages more than one another webservice to establish its deliverables. By loosely coupled, a webservice can be developed without knowing its consumers, and new consumers can be on-boarded without modifying the webservice. A service can be composed of one or more other webservices which leads to complexity in engineering the webservice. There are several challenges at design phase of webservices including designing communication tools between service provider and service requester, specifying service level agreements with or without prior knowledge of nature of consumers, planning a security enforcements, etc. Service-oriented architecture (SOA) is widely adopted software design paradigm for webservices and addresses some of the challenges in webservices development process. The development life cycle of a system of webservices involves identifying services, isolating them from one another, governance of services, distribution of service development tasks, maintenance. Conventional software engineering principles become invalid for webservice development and tailor-made engineering approach called Web Service Engineering (WSE) is brought up for webservices. This chapter presents nuts and bolts needed for WSE. WSE borrows concepts from agile-based Software Development Life Cycle (SDLC) for webservices to call it Web Services Development Life Cycle (WSDLC). In WSDLC, a webservice can be developed independently from other services in an enterprise, with complete isolation from other services in the enterprise. Internet era is reaping benefits from majority of webservices available across the globe, a streamlined WSDLC approach helps several industry partners to coordinate each other in a systematic language. By the end of this chapter, a reader should be capable of developing a webservice in WSDLC. Webservices revolutionized software systems in industry and few of the organizations that reaped benefits from webservices approach are discussed in the chapter as case studies.

V. S. Martha (✉) · M. Lenglart
Lore IO, 380 Altair Way, Sunnyvale, CA 94085, USA
e-mail: venkat.taku@gmail.com

M. Lenglart
e-mail: maurin.lenglart@gmail.com

H. Mohanty and P. K. Pattnaik (eds.), *Webservices*,
https://doi.org/10.1007/978-981-13-3224-1_7

Keywords SDLC · Webservices engineering
Webservices development life cycle · SOA · Microservices

1 Introduction

Internet era made many software accessible over web as web applications. A software is typically a combination of one or more modules and similarly a web application is a collection of one or more modules. In recent times, the modules evolved into "services" and further the services are designed to serve over web to call them "Webservices". Webservices are derived from web applications, where web application presents data in a user interface and a webservice only serves data. Briefly, a webservice is an autonomous software component designed to serve a specific functionality or a task over web. A service provider and a consumer interact over web in webservices framework. A module in a software can be a webservice if the module is accessible over web. Typically, web end points, also called Universal Resource Locators (URLs), serve business functionality over web, and therefore each URL is a webservice. Citing an URL as a webservice is more general, various groups in research community defined webservices differently. IBM defined a webservice as "a self-contained web application that is published, accessible over web". Everything accessible over web is not a webservice but functional units. A webservice is an artifact with a published static programming interface and not a set of web pages. As in client and server in web application, there are service provider and service requestor in webservicer. Service provider owns a service, while service requestor consumes the services from service providers.

Service providers publish webservices to a registry and a service requester finds a service and invokes the service over web. The interactions among service provider, service registry and service requester are regulated by a contract which all the parties agree upon. There are popularly two types of contracts, Simple Object Access Protocol (SOAP) is one and REpresentational State Transfer (REST) is another one. In SOAP, a webservice definition is published in a Webservice Description Language (WSDL). WSDL is an XML document for a service that describes the service. A service requestor communicates using SOAP messages with a service provider as specified in the WSDL for the service. A SOAP message is an envelop of a header and a body, where the header contains instructions for reading and processing the body of the message. On the other hand, a REST service is accessible via a Uniform Resource Identifier (URI) and avails a set of resources through a set of interactions. The interactions are well-defined operations on the corresponding resources. PUT, GET, POST, and DELETE are the operations.

As webservices taking vital role on the Internet, there is a pressing need to enforce systematic, disciplined, and quantifiable approaches to development, operation, and maintenance. Web engineering principles, largely borrowed from software engineering, are brought up to systematic and disciplined approaches to effective life cycle of the web applications. Web engineering community has been working on issues and

challenges involved in web development process. In addition to traditional software engineering principles, web engineering outlines concern from open and flexible nature of web. Primarily, there are two factors that make webservices different from typical software. 1. Frequent updates to the underlying data, 2. Continuously changing requirements from diverse stakeholders. Both the factors make a webservice endlessly evolving in nature and webservice should be engineered to support such evolutions over time. Success of a webservice is dependent on ability to extend, scale as with its stakeholders. According to 2015 CHAOS report published by Standish Group, only 29% software projects are successful. With the reach of webservices, a failed webservice can lead to a crisis, often called web crisis.

Effective engineering principles needed to achieve successful web applications and web engineering deals with the process of developing web applications. Web engineering provides a superset of principles for overall web applications and webservice is part of the web application.

As each resource in a web application is available as a webservice, it is common to have a webservice that interacts with one other webservice to access a resource. A webservice that relies on other webservices is called "composite" webservice. Webservices often interact with more than one webservices and form a web of webservices. A web-based system becomes complex when the size of web of webservices abscess though the system is modularized with each service as a module. Service-Oriented Computing (SOC) is the term coined to represent the applications developed using a set of services. A system is architected using Service-Oriented Architecture (SOA) to streamline the development process when the system is comprised of services. SOA is a typical way to develop webservices. The terms "SOA" and "Webservices" are used interchangeably to represent a suite of services. SOA helps development teams to consistently deliver sustainable business value, with increased agility and cost-effectiveness, in line with changing business needs. In SOA, each is a loosely coupled isolated component with reliance on shared resources. SOA lets a webservice reuse other services and resources without worrying how the services work. A webservice plays "consumer" role to consume another webservice, thus a webservice is a producer and a consumer at the same time.

Service-Oriented Architecture (SOA) is a software design to manage the usage of services in terms of, and in sets of, related services. The World Wide Web Consortium (W3C) defines SOA as "*A set of components which can be invoked, and whose interface descriptions can be published and discovered*". SOA like software design pattern will have significant impact on the software development life cycle of an application. SOA is not just an architecture, but the policies, practices, and frameworks to ensure the defined service interaction. SOA enables the development process to follow independent development process for each of the service in the application. An application may portray many to many relationships among services; the relationships are regulated by service contracts. Consumers request a service with different objectives but nevertheless use the same service. A service is to be developed independent of other services in the application but the development should comply the contracts with its consumers who can also be part of the application. Such an independent development with dependent requirements (contracts)

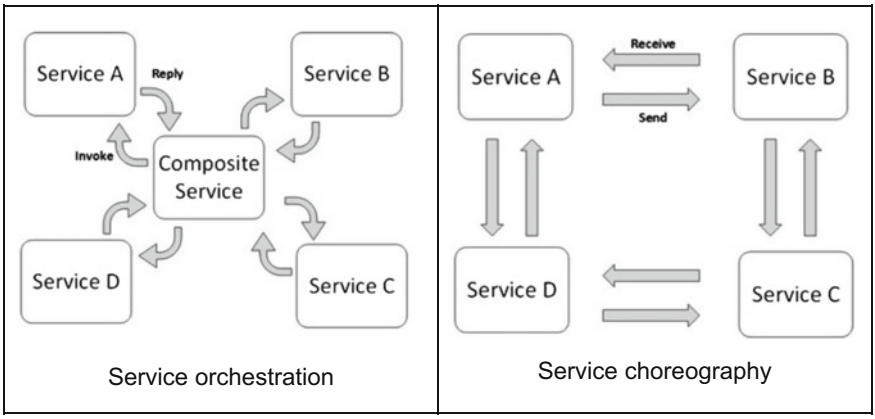


Fig. 1 Service composition mechanisms

demands a specially orchestrated development life cycle and this chapter discusses an extension of traditional software development life cycle to support webservice development.

SOA is suitable for webservices where

- Services operate over the Internet;
- Every service requestor adheres to changes in provider;
- Need for heterogeneity; and
- To make use of a legacy system in a technology that is not the legacy system support. To wrap the legacy system in a webservice.

A webservice can consume other webservices, accomplishing a dependency on the other service. It can also be viewed as a composite service which invokes one or more dependent webservices to trigger other business processes. A composite service is not responsible for the functionality of the dependent services but to trigger them as per demand. There are two types of service dependencies possible in SOA, one being "Service orchestration" and other is "Service choreography".

1. **Service orchestration**: A service is orchestrated with a composition of other services and the service is called "orchestrator" or "composite service". The orchestrator is the central point to engage with other services. No communication takes place among other services pertaining to the orchestrator's service. Only the orchestrator is aware of the goal, so the orchestration is centralized with specific instructions to when and what services to invoke.
2. **Service choreography**: The services are decentralized so that a service provider can request other services for a service in realizing a service request. This approach enables the services coordinate themselves with none of them having control over other services. Each service is responsible for its only functionality and serves its consumers effectively (Fig. 1).

Choreography is a collaboration while orchestration is a centralized execution process. There are pros and cons with both the approaches and both pose challenges in their respective ways. In both the approaches, services are distributed and engaged over web, inherently infusing challenges in distributed systems into SOA. And, webservices adhere to client–server model with service requestor being client and service provider being server, thereby inheriting problems associated with the model. The challenges involved in engineering webservices are discussed further in Sect. 2.

2 Challenges in Engineering Webservices

One of the objectives of SOA is the identification, development, deployment, and life cycle management of services (webservices). Webservices development process depends on how a webservice would be used. For example, a webservice has to be engineered by taking account for how a requestor requests a service, which protocol to be used, what to expect from the service, and semantics of the request to be placed. The challenge is in preparing specifications for the service without prior knowledge of who and how to consume the webservice. Discussion on some of the challenges is following.

2.1 Service Registration

Webservices are registered with registry to let its consumers know about the service. Now a question is, what details are to be registered. The registration process requires the service provider to provide what is the service about and how it is accessible to consumers. Following are the metadata needed for a service registration.

2.1.1 Messaging Language

Service provider be able to specify a common language which can be used for communications. The language is used to specify service description. XML is widely accepted for this purpose.

2.1.2 Interface Language

Interfaces define how to consume a service, it is the service description. Typically, Webservice Description Language (WSDL) defines the interface of a service and is submitted to registry at the time of publishing a service. For resource-based webservices, Universal Resource Identifier (URI) of a service and protocol to reach the URI present the interface of the service.

2.1.3 Business Protocols

A list of operations that a webservice offer is described as business protocols. Business protocols help a consumer to communicate with the service provider to perform a function. Business protocols are rules to validate which operations are allowed and when. GET, PUT, POST, and DELETE are business protocols for REST-based webservices. Web Services Conversational Language (WSCL) and Business Process Execution Language for Webservices (BPEL) are examples for business protocols.

2.1.4 Properties

Service provider can submit a set of properties associated with a webservice. The properties can be for service level agreements, QoS, license details, etc. The properties complement the information from interface language by adding a layer of information that is not associated with functionality of the service.

2.1.5 Verticals

Vertical standards define a specific common base language, interface, business protocol, and properties. This is a compilation of metadata of metadata.

A webservice development is tightly tied to metadata of a service, lays foundation for the service development. Selecting appropriate interface, business protocols will be game changer for the service throughout its life. The challenge here is to make decisions on the metadata with prior thoughts on the lifetime of the service. Once a selection is made for the metadata, it is nearly impossible to change, even if it can be changed, such migration would cost significant resources.

2.2 Service Binding

A consumer consumes a service by connecting to the corresponding server found in service discovery phase. Binding mechanism varies with the protocol used for the service accessibility. Resource-based webservices are transported over HTTP or HTTPS and object-based webservices are accessed over SOAP messages. A webservice development process depends on the binding model chosen for the service.

2.3 Service Composition

If a service is composed of more than one service, the service composition can be through orchestration or choreography. Though both the compositions are backed by

benefits and limitations, only one type of composition is to be adopted. Based on the context and requirements of the service, a service is composed. No matter what composition a service is, for a consumer it is quite similar to typical webservice and does not observe differences. Service provider needs to make respective developments to make the services compatible with the chosen composition.

When a service is composed of services from different enterprises, there is a need to standardize on the relationships among the enterprises involved in the composition. The goal of each relationship of a pair of service provider and service consumer is to produce and consume. WS coordination specifications are compiled to detail how a webservice can be accessible to other services. WS coordination specifications help to achieve interoperability among services, the scope spans from a service within an enterprise, across different services in an enterprise, across enterprises.

2.4 Integrity of Services

Since a webservice can access a shared resource or other services, it is possible that the consistency for the service is not guaranteed. A shared service can be requested by more than one service, and if the resource behind the requested service is updated by the first service request, the subsequent service request is possibly working on latest data with instructions from outdated data interface. For example, a service that access "Employee" table is called to raise salary of 1000$. If a service is called twice for an employee "X" when the salary was 5000$, the service invokes one request to increment salary to 2000$ and subsequent request is operation on "ahead" data than the request was invoked to raise the salary to reach 7000$, which is non-intended.

2.5 Governance

The webservices are loosely coupled and engage with one another to form a composition. More than one service can consume a shared resource and a webservice can be dependent on functionality of another service on a resource, thus making a service dependent on another service without consuming it but indirectly from the shared service. Such complex relationships among webservices are possible in SOA. More the number of webservices in a system more the chances of encountering dependency issues. A strategy is needed to govern how the services are developed, deployed, and managed throughout their life cycle to accomplish enterprise-level suite of webservices. SOA governance is to be put in place as part of engineering the webservices, to ensure every webservice is given requirements with its functionality, what to consume from other services and how the service is contributing to overall business goals.

2.6 Availability and Performance

As webservices are highly distributed across web, the performance goals can be a challenge. Moreover, mission critical system started adopting webservices, and it is of the highest importance to have the services available and perform as per requirements. A service provider should enlist what a service requestor should expect from the provider, which is called Web Service Level Agreement (WSLA). WSLAs are contracts between a service provider and a requestor to agree on the obligations of both the parties. In webservices terms, WSLA is a commitment from a service provider to fulfill a service according to agreed upon guarantees. A WSLA can include guarantee on availability, response time, throughput, etc. For example, a WSLA for a webservice, to serve time stamp, includes the service that never have a downtime more than 10 s without a week's notice, 10 s response time. With the WSLA, a service requestor can develop a system with the commitments into consideration. For instance, assume a service requestor is consuming the webservice, this requestor can complete his task no less than 10 s, and the limitation inherited from the dependent webservice.

2.7 Security

The webservices are put on the Internet to make it available over web. Since web is open to everyone, it is necessary to enforce security policies to limit access to the service. Typically, authentication and authorizations are services itself and leveraged all services in a web application. Each request to a webservice invokes a request to the authentication service to authenticate the request and another request to authorize the request. Once a request is through, the request starts to be fulfilled by service provider. Who can access a webservice is to be well prepared for the deployment of a service. The webservices that provide security-based functionality are vital in enterprises that operate on sensitive information. WS-security is a webservice specification to enforce integrity and confidentiality to webservice communications. When a webservice is designated with a security protocol as specified in its WS-security specification, the webservice is accessible to only who adhere to the protocol.

3 Software Development Life Cycle for Webservices (WSDLC)

Concepts from software engineering principles put together to address challenges mentioned in Sect. 1. SDLC is heart of software engineering and there are several SDLC models, each of the SDLC models has benefits and drawbacks. SDLC for webservices will be discussed following a brief introduction to SDLC models.

Typical software development life cycle methodology involves five phases: 1. Requirements gathering, 2. Analysis and Design, 3. Implementation, 4. Test, and 5. Release/Deployment. A methodology to stitch the phases to deliberate a software development is called software development process, and there are various methodologies existing in industry.

1. **Waterfall model**: Waterfall model is a naive software development methodology where the development phases are executed in sequence. Software development starts with requirements gathering phase and ends with deployment. There is a maintenance phase after deployment which continues till the end of the life of the software. A phase is performed one and only once.
2. **Iterative and/or Incremental model**: Waterfall model is performed iteratively and with every iteration releases a version of the software. Software evolves over iterations; a development phase is performed more than once as many times as the number of iterations. If a feature is not fit in an iteration, it will be added to next iteration and will be released in the next version of software. It is also possible that the requirements for the software also change over time and the new requirements will be considered to fulfill in the next iteration.
3. **Spiral development process**: Giant software systems require significant preparatory work before going onto development and prototyping is a way to verify the feasibility of the designed system. Spiral model enables to deliver prototypes at one or more iterations. Once a prototype is evolved to satisfy the requested requirements, the rest of the development phases progress in sequence to deliver the software at the end.
4. **Agile process**: Software is divided into one or more modules (can be called features) and each module development is carried out by a team. Such a modularization reduces the effort in planning and development of complex software systems. A software module is developed iteratively and involves cross-functional teams and each team takes part in all the phases of the development process. A version of the software is released at scheduled time points and the release may not include all the software modules. Each team sign off the corresponding module; the modules are integrated to form the complete software that goes into release. Though each team is responsible for its module, there may be dependent modules to impact each other's development process. Cross-functional teams communication is essential in such software development (Fig. 2).

None of the listed SDLC models here can suite for webservices development as is, because webservices are loosely coupled with isolation. A set of webservices can be developed in parallel as there are phases of webservices development that can be distributed and optimized. Webservices are distributed artifacts and each webservice is associated with its specific requirements that enable it to follow development on its own path.

Webservices are similar to independent software modules in agile process but unlike agile model, webservices are heterogeneous and follow independent development process. Webservices development process borrows many of the features

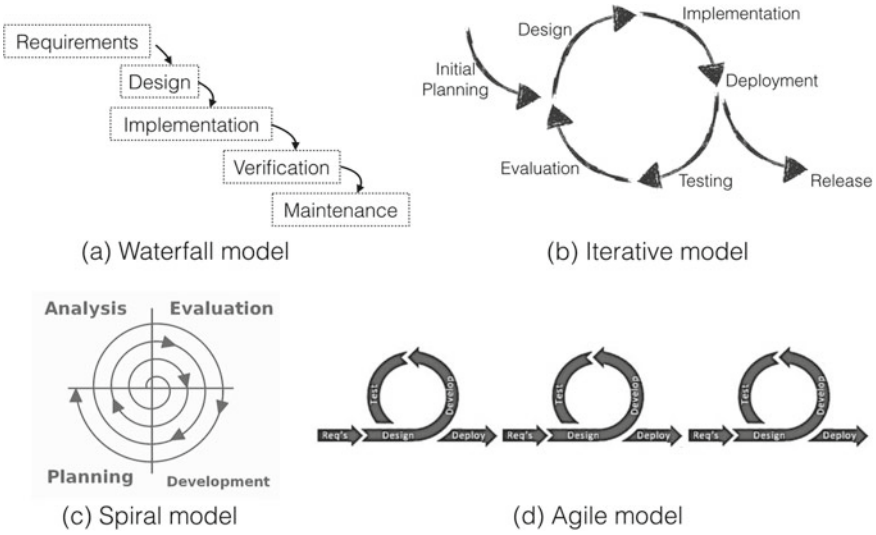


Fig. 2 Flow of development phases in traditional development methods

from agile development process. Further, illustration of the webservice development process is presented in the following section.

A webservice in a software system can be completely disintegrated from the system and follow its own path in development. Therefore, a software system that is constituted of webservices is developed in different ways but put together to make up a software solution. The development process in such software solution is heterogeneous so that it is complex to engineer a process for the development. Conventional development schemes discussed in the previous section cannot fulfill the needs in webservices development and this section addresses the need for defining a development model specifically for webservices.

As mentioned, a webservice is an independent software module that can serve its consumers independently for the specific functionality it is designed for. A software system can be made up of one or more webservices and there can be many-many relationships among services in addition to relationship with consumers. The consumers interface with a webservice for a functionality and the service can utilize other services in accomplishing the requested service. Typically, a webservice that uses another webservice plays a "consumer" role to consume the service and there a webservice can be a consumer of another service while service process for its functionality. Such scenarios result in service dependencies (contracts) and dependencies are not uncommon in system of webservices. Though a webservice is dependent on other services, the webservice is isolated from the dependent services and acts independently. Therefore, a webservice is an independent software module with dependent services. A pictorial representation of a software system of webservices is depicted in Fig. 3.

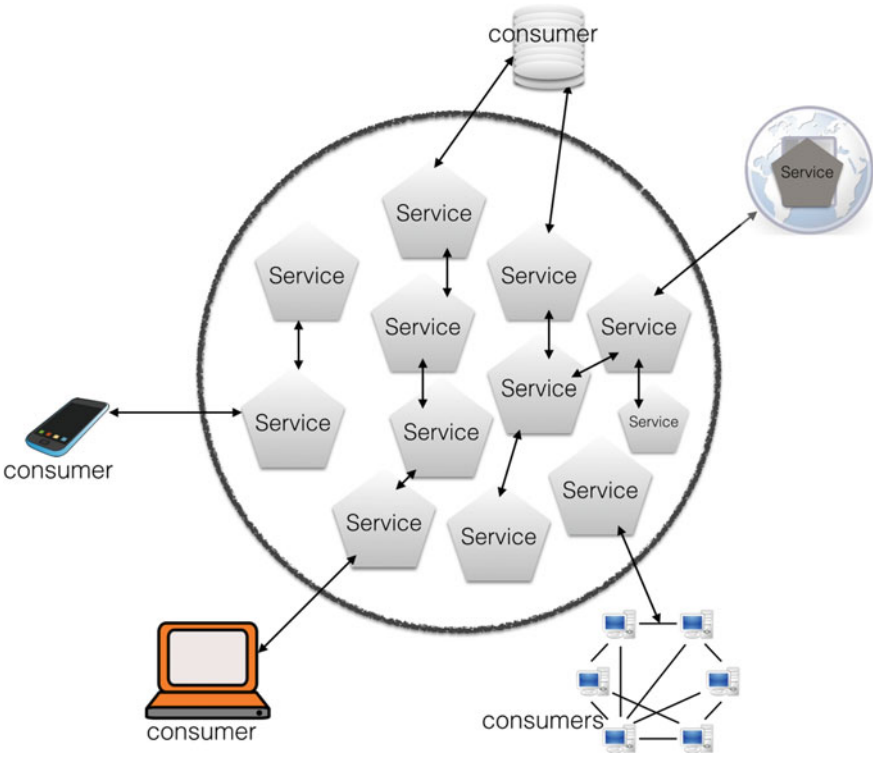


Fig. 3 Webservices architecture

In order to keep focus on the development process, further discussion on the webservices architecture has been skipped here and assuming the reader having ample prior knowledge of webservices architecture.

There is not enough research made into standardizing the development process for webservices and this chapter is an attempt toward the direction of instrumenting the artifacts needed for webservices development model. Though most of the problems that arise in webservices development have been addressed in literature in different contexts, a development life cycle model for webservices is not mentioned in the past. By calling out to have a comprehensive development model, this chapter paves a way to combine the available solutions for problems in webservices development. A webservice development model can be seen as an extension to agile model as webservices possess agility in heterogeneous environment. Webservices are no different from typical software modules but with complete isolation from rest of the software modules, and therefore webservice development process borrows several development phases from the traditional software development models. Elaborated illustration of each phase in webservices development process is following.

3.1 Requirements

Requirements gathering is the first and foremost step in a software life cycle and web-services-based systems do follow the same tradition of collecting all the requirements of functionality. The requirements list defines what the system of webservices should serve its consumers.

Requirements collection starts with identifying stakeholders of the software. Stakeholder(s) can be defined as customers, end users, a project manager, etc. A comprehensive list of requirements is prepared by meeting with each of the stakeholders. The outcome from the meetings is requirements document which will be passed onto next phase in the development life cycle. The document also serves as contract with consumers on what the software is guaranteed to offer.

3.2 Design and Architecture

The requirement document from the previous phase is the basis for planning the rest of the phases in development life cycle.

In typical software development life cycle, this phase translates the requirements into a design document which will be used to code the software. Unlike other software development models, webservice development process uses the requirements document to decouple the requirements into one or more requirements documents. Each requirement document serves as a basis for a webservice so there is a webservice for every requirement document that came out of this phase. Briefly, this phase architects the software in terms of webservices. The only restriction on splitting a requirements document into many documents is each requirements documents should serve an end-to-end solution for a specific functionality. By making each requirements document an end-to-end solution, each webservice is isolated from others and developed independently.

Webservice governance is orchestrated in this phase to coordinate services involved here. Each service made here is responsible to fulfill its requirements document and does not have overall requirements in the goals. A democratic approach distributes overall goals and how each service contributes to overall goals to every service. Though each service is aware of birdview of the system, a service is just focused on its specific functionality scripted in it requirements document.

Webservices are secured using common rules and in this phase the rules are made out. The derived services from this phase follow the security guidelines prepared in this phase. Often times, the security guidelines are programmed as a service and all the derived services leverage the service to adhere to common security policies.

The requirements documents are distributed among engineering teams for the development where an engineering team is capable of performing all the development phases in traditional development life cycle for the designated webservice. Given

the need for a team that is capable of complete development, typically such teams constitute of individuals with design, implementation, testing, and release skill sets.

It is not unusual to receive new requirements from stakeholders and subsequently this phase is reinstated either to infuse the new requirements into existing web servers or to introduce new requirements document for an addition webservice. As the number of webservices increases in the system, the overall software evolves increment. This process is widely popular in small-to-medium scale software products by releasing a minimum viable product with key webservices and then adding additional webservices to extend the software for wider consumer requirements.

3.3 Webservices Development

A development life cycle of a webservice starts with this phase as previous phases only drove the process to define webservices. Being each webservice is isolated from other services, the development process for the webservice can be one from many existing software development models.

Regardless of what methodology a webservice adapts for development, the development phases are common among all models but the phases are wired differently to achieve the webservice releases at different rates. The development phases for a webservice are discussed here to give the reader a complete overview of the development process and the discussion is analogous to typical software development models.

Each team responsible for a webservice acts independently on their development schedule and relish the freedom of choosing a development methodology for the owned service. Figure 4 depicts one such organization structure where services are developed using different schemes. A service follows waterfall model as in Fig. 4a, one service in iterative development methodology as shown in Fig. 4b, spiral model (prototyping process) is adapted for a service as in Fig. 4e, where two services are following agile process as in Fig. 4c, d. In an outline, Fig. 4 demonstrated the capability of webservices development process to develop independently in a dependent services system.

3.3.1 Requirements

This stage of the cycle is used to take all the requirements and translate them into features and technical specifications. It can also include screen layouts, mocks, and pseudocode. But also how the webservice is supposed to be architected. The requirements also include all the stakeholders for the webservice and corresponding contracts with them.

Each of those requirements needs will be categorized into two main categories as follows:

- Functional requirements such as features, business rules, authorizations, etc.

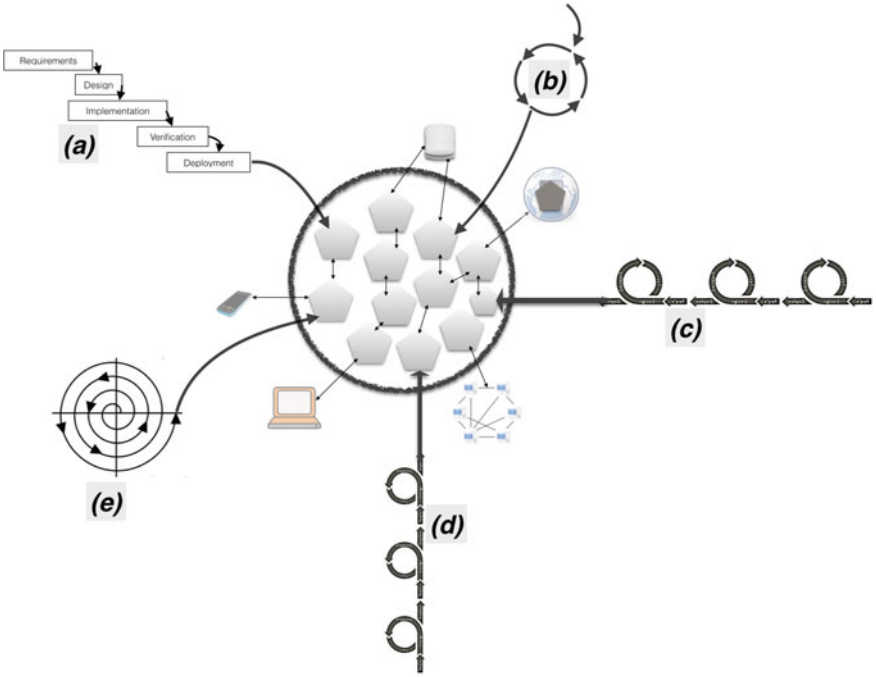


Fig. 4 Development of services in web-services-based system

- Nonfunctional requirements such as usability, performance, scalability, supportability, data integrity, etc.

A webservice is supposed to only solve a fixed amount of problems. So it is important to, once the requirements are analyzed, make sure that a webservice can solve it and that it is the best solution possible. During the development of a webservice, the requirements can, and mostly will, change over time. Since the changes in requirements are tracked regularly, the requirements are needed to re-evaluated upon changes.

Further, requirements document includes how consumers of a webservice consume the service. How the consumers use the webservice will drive us to construct a test plan for validation of the service, and thus can verify the given requirements being achieved upon a release.

3.3.2 Design

Given a requirements document, the webservice is designed to satisfy the given requirements and achieve the functionality goals designated for the service. As a first step, a signature of the webservice is prepared where the signature defines how

a consumer communicates with the service and what to expect in response. In simple words, a signature is a schema of input and output data for a webservice.

Once the inputs and outputs of a webservice are identified, the functionality of the webservice is orchestrated into a design document. The design document includes architecture diagram, pseudocode, dependency resolution, system impacts, the list of requirements that are being achieved and anything that needed for the implementation of the webservice.

Once the requirements are documented, following design choices has to be made: 1. an interface to use, 2. a messaging language to use, 3. list of operations to support, 4. orchestration or choreography, 5. integrity of service, and 6. SLA. The choices made for these will define the webservice accessibility. The functionality of service is designed by the choices made.

When a design phase identifies a dependency on some other webservice, the dependency is created as a requirements document and the requirements document is spin-off to a webservice development phase (C) and the process repeats for the webservice. Often times there is an existing webservice that satisfies the requirements of dependent webservice and no new development cycle is introduced but reused existing webservice.

3.3.3 Development

The development is where the code is being written. The goal is to take the design document and translate the algorithms and the pseudocode created in the design phase are translated into a programming language.

Typical software system is developed by reusing existing software modules and does not need to implement end-to-end solution all by itself. Since a webservice is an end-to-end solution for a functionality, the development team develops all needed tools except the ones available via other webservices. Thus, the development process can consume more resources but helps in keeping the webservice isolated from the rest of the system and to enable independent development cycle.

3.3.4 Testing

Borrowing the lines from the requirements document that illustrate how a consumer uses a webservice, a test plan is written for the webservice. The test plan is validation of the webservice onto check whether the given requirements are met. Being a webservice an end-to-end solution for a functionality, the tests can include black box testing. This is the phase where other webservices that consume this service can verify the contract. Since a webservice is serving a specific functionality, the testing plan is succinct and less resource consuming.

3.3.5 Deployment/Release

The release schedule for a webservice is defined by itself disregard of the overall release schedule. There would be versioning in the release to flag what version of webservices should go together to end consumer. Such versioning helps to serve a consumer a service derived from all specific versions of services, and thus maintaining consistency in service. The versioning is not always needed but depends on the impact of the release on other services and the services that reach end consumer.

Irrespective of the versioning/flags, deployment of a webservice would be simple as it does not need to bring down overall software except the webservice in critical path in consumer services. If there are services that consume a service and had to make a release, then both the services have to go together if there is no versioning. There are companies that deploy releases few 100 times a day and not all of them available to end consumers right away but checked with a flag to make sure that the consumer avails only stable version of the service.

There is no concept of integration in webservices but the webservices consume one another and consistency in the contract guarantees the integration does not fail. There will be a regular review on overall webservices at very high level without bringing in service details into review discussions. Thus, the review meetings oversee high-level architecture, while inside each engineering team there is an evaluation phase to review low-level details of a webservice. Webservices have been evolving and many enhancements have been made to the webservices to address problems over time. Some of the advancements are discussed in next section.

4 Advanced Techniques in WSDLC

Webservices have been in the industry for more than a decade, over time webservices evolved into sophistication. With every new advancements in webservices, the development process has been adopting to the newly introducing concepts. Some of the cutting-edge technologies in webservices are discussed here to establish their impact on the development process.

A. **Microservices**: With growing size of software systems in organizations, webservices are being further microfied into smaller software pieces. Such microsoftware pieces are called microservices and the microservices feature more benefits in addition to being smaller in size. By exhibiting more agility, microservices became popular among the webservices. Microservices are not just isolating software components but adopting stringent practices in development life cycle. Besides development process, decoupling the services in terms of microservices needs more effort as the microservices facilitate more independence and capability. In the long term, the process makes the development lucrative with faster iterations and releases. Like webservices, microservices also developed independently but with more contained implementation. Since microservices are

smaller, it takes little effort to develop and maintain each service. At the same time, the integration could become burdensome as more number of pieces stitch together to make an overall system release.
But by opposition to webservice, a microservice can be bundled into a "package": npm module, python egg, jar file, etc. allowing to be directly integrated into an other software. It reduces the performance problem encountered.

B. **Continuous development and integration**: Conventional software development process involves sequential phases of development life cycle, while agile methodology allows to develop parallel development of modules in a software system. With the evolution of advanced technologies in webservices, it is possible to not only develop in parallel but deliver parallelly, in other words, deliver continuously. As in agile methodology, each webservice is implemented in parallel and then integrated at regular intervals without needing to know the progress of other services as long as the delivering feature does not impact others. Thus, a feature can still be delivered irrespective of some or more of other features delivery. In this process, the integration is critical phase and rigorous testing following the integration involves every team that integrated their implementations. As more than one team integrates their code near real time, conflicts in the implementation arise. The only way to reduce the conflicts is to communicate the changes beforehand to each team that interact with the service. There is a version of the product with all the integrations made and where the testing team runs their test cases. The integrated version of the product goes back and forth with development team for issues arising from integration. During the testing phase, the development process still continues onto the next set of enhancements of their respective services. Once testing team concludes the testing, the system goes into deployment phase. The process starting from integration to deployment is independent of development of services, and therefore continuous development and deployment. It is not uncommon to see unfinished features that drove themselves into integration phase and such features are put under flag to hide from customers. Facebook uses similar strategy that deploys unfinished features into production system and there is a tool called "Gatekeeper" to manage what feature a user can see on the Facebook web portal. Keeping a feature off track from customer usage buys auxiliary time to enhance the feature. At times, the gatekeeping methodology is exploited to achieve incremental deployment.

C. **2-Pizza teams**: Witnessed success at Amazon, 2-Pizza teams strategy has been popularly adopted across industry. If an engineering team, owning a service, can eat more than two pizzas, then the service is too large to achieve agility and the service is eligible to be broken further to make it fit into an engineering team. The engineering teams are separated into silos and operate independently. In web-services-based industry, a 2-Pizza team is responsible for a service, and therefore the services should be small enough to be developed by such 2-Pizza team independently. If anything goes wrong anywhere in that full life cycle of the service, these 2-Pizza teams are the ones accountable for fixing it. The teams comply with SLAs internally and thus services communicate with each other without conflicts.

D. **Containerization**: Microservices and containers ride hand in hand as one benefits the other. Containers enable each service to run independently in a heterogeneous environment. A service is free to choose its specific programming language, an operating system, or a hardware without worrying how the rest of the system is being designed. Succinctly, a service is a virtual machine serving its consumers a specific service. Containers are lightweight processes emulating a virtual machine and capable of bringing up given system environment. A physical computer can support more than one container at an instance and each of the containers in a machine is isolated from others. Nowadays, the cloud provider is offering containers out of the box to deploy services. If not on cloud, containers are deployed on dedicated machines. Google managed billions of containers internally for years and added Google Container Engine (GCE) to its infrastructure, Amazon provides quick way to deploy containers on its EC2 cloud, and Microsoft added container support with windows server containers. Containerization evolved to become de facto standard for microservices.
E. **Serverless services**: Since the services functionality is shrinking to microlevel, it is not feasible to operate a machine or container for each service. In order to bring down operational cost, the services are deployed on a server that runs the deployments with isolation. Services can take advantage of abstraction layer from the infrastructure and developers can focus on the service functionality without distracting into infrastructure, scalability, or security. The solutions developed within a system for infrastructure, scalability, and other components will be available for all the services and the services take benefit from the common solution in the system. Serverless services are referred in general as "Function as a Service (FaaS)". Though we call it serverless services, the services are served from a common server for all services in the system. Amazon offers serverless services platform on AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions. The services developed for a platform will be based on the modules offered by the platform, and thus it is burdensome to move a service from one platform to another. Such a lock into a platform denounces one of the fundamental properties of webservices. Serverless computing is rapidly evolving and it could be possible to have a standardization to ease service movements.

Over the years, webservices went through revolutionary transformations and have been evolving steadily.

5 Challenges in WSDP

Though webservices approach solves many development process problems, there exist scenarios where webservices do not fare well. At times, adopting webservices incur challenges over course of development process. Some of the challenges are listed as follows:

1. In case of applications with tightly coupled functional units, decoupling become difficult and could end up with nonuniform services. An application development is seamless when the application is divided into several services of nearly equal development resource requirements. Such nearly symmetric division helps in scheduling continuous integration and delivery. It is not always possible to divide services uniformly, but recommended to dedicate best effort toward the uniformity.
2. There are two types of decompositions possible and each of them comes with pros and cons. One of them is "Verb-based" in which each service is a customer use case. Other type is "Noun-based" where a service represents an entity and associated with all operation on the entity. Based on the application, one can choose either or both the mechanisms.
3. A number of webservices grow in an organization over time and monitoring all the services is burdensome. Webservices are typically committed to always-on SLA, and supposed to be available at all times. Such a concrete agreement makes the development process to consider quality testing toward the availability in addition to functional and performance testing. Besides testing, the maintenance of the service also impacts the development life cycle.
4. A webservice can consume one or more other services in the application and thus the services has to agree on the way they exchange information. The development teams have to communicate on regular basis to comply with the agreements. A team responsible for a webservice has to bring in all of its consuming webservices to update them on the changes planning in the service so that the other services reshape their service accordingly. The meetings to bring in teams on regular basis is taxing in development process and could slow down the process given the meetings could change the course of the webservice.
5. The cost of running more than one service is resource consuming compared to monolithic application. As each service runs independently in an isolated environment, altogether consume more resources than a monolithic system. Such high up-front cost is not desirable for smaller scale industry but can benefit the team to scale as they grow.
6. Performance is one of the hurting challenges. Since services are itself consuming one or more other services and thus moving data, computing control from one service to other to increase latency in the overall service response. It is also possible that a service in the application could be bottleneck and impact the services that depend on it. In addition to data movement across services, while moving data from one service to another there are possible data transformations happening before and after the service send and receives data. Such transformations again incur resource consuming and reduce performance of services.
7. Since a webservice is focused on serving a very minimal functional unit of the application, it is not possible to embed security features into the webservices development and so webservices are prone to vulnerabilities. The webservices have to be secured to enable interservice communication flawlessly and typically

security is another service in the application consumed by other services for secured services.

8. Debugging become harder, as any service can be a point of failure. If a software fail to behave like expected, any of the dependent service can potentially be the root cause of a bug. Just understanding where a problem comes from became a really challenging task.
9. Each service can potentially be developed in a different language with different frameworks and different code styles. It can tie a developer or a team of developers to a particular service and limit their knowledge and understanding of the overall architecture.
10. Each team is contained to one or a few number of services. It then becomes really challenging to build a feature that will require the need of modifying multiple services across a company.

6 Case Study

Webservices became de facto standard in software industry and many organizations ranging from small to large scale are incorporating webservices into systems. A bite from some organization known to author discussed in this section to construe the webservice development process. Author is no way soliciting for the organizations chosen in this discussion but calling out the experiences in the organizations to study the webservices.

Google

Google has been providing wide range of services for the better human living. The services range from navigation to weather, humanitarian assistance to business process, and marketing to e-commerce. Though some Google services incur cost, there are plenty of services that are available at no cost for noncommercial use. Not just its partners but many individuals have been developing many applications based on Google's webservices. Such applications consume the always-on services provided by Google to deliver the applications' goals. It is not uncommon to keep under the carpet the details of the service implementation and Google owns responsibly of the services all by itself. Given such an independence to services, Google can enhance a service with nearly no dependence on its consumers or consumer goals.

On the other hand, an application depending on a service may need a plan to integrate the service into its development process. Typically, Google services follow REST interface for its services where each service is treated as a web resource. A service is associated with a web URI through which a consumer requests a service. Google server responds to a requested URI with a response respective to the parameters passed in addition to the URI. Google webservices support many platforms and various programming interfaces. The way an application interacts with a Google webservice may differentiate the development process.

From the initial product idea, a product manager prepares requirements documents, often times the requirements is a mockup. The features in the mockup are divided into several pieces, call it services. Most of the times existing Google services can satisfy the requirements and very few new services needed to be implemented. An engineering team, including a team lead, is assigned to each new service to be implemented and plan the service development to achieve the given requirements. The engineering team is of very small size, about 3–10 engineers, and the service assigned to them to be small enough for the team to deliver at stipulated time intervals. There will not be a release deadline for the product while each service develops at its own pace and delivers at custom timelines. At high level, the product manager meets with a team representative, usually called team lead, from each service on regular basis to review the goals. It is not unusual to modify the goals of a service in such review meetings and such scenarios put the development of the specific service incur critical issues. At times, the service with new goals is to be started on from the scratch by winding the progress at the time point. Such a massive overturn of the service may not impact the overall product development, it is possible because of agility of the service-based development process.

Unlike agile development, services (features) in Google are released incrementally. As long as the service does not meet the minimum viable feature request in the product, the service is put under a flag to hide the feature from customers. The process similar to circular train, in which you fill bogie of a train at your stop on regular interval, and the bogie spun out to attach to a train with a destination when the bogie is full. Similarly, the service is released incrementally at regular intervals and the product carries the feature under a flag until it is ready for the customers to consume. Once the service is a full-fledged feature, the flag is lifted out to make it available for consumers. A product release goes on scheduled time despite some of the features/services that are not ready. The services that could not make it into a release will be scheduled as part of the subsequent release. In summary, Google believes in "Release Early, Release Often" strategy.

Such a process helps to continuously deliver new features into the product and keeps engineering team's focus on specific goals despite the gigantic product in whole. There is a long "feature freeze" period for testing, translation, and stabilization of a service. The release team makes a release every time period, typically 6 weeks, and then the development team just keeps progressing the development process (Fig. 5).

In this process, there are three live versions of the software at any given time—the production release, the QA version that is in stabilization, and the development version. Operations teams find it cumbersome to have three completely separate environments with databases, etc. (development, QA, and production) for each webservice. For the reason, Google started utilizing lightweight virtualization mechanism called containerization. Each webservice can be contained with its specific environment configuration. Popular Google services such as Gmail, Search, Apps, and Maps run inside containers. Google, which deals with more than 2 billion containers per week. That is a lot of containers to manage without automation tools. Google has been building several software packages to deal with such containers at billions in num-

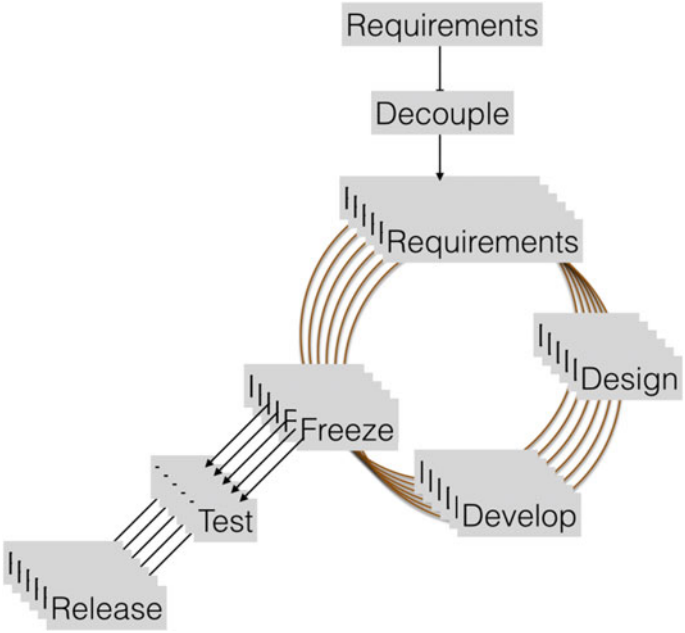


Fig. 5 Webservices development process in Google

ber. One of the first tools that Google released to for public availability is called "*Kubernetes*".

In summary, Google has been investing resources to facilitate services development with nominal effort, also moving toward microservices approach. The services are being made into completely independent including platform, data, etc. By leveraging service-based development process, Google has been able to optimize resource utilization and continuously deliver at faster pace.

Amazon

Amazon is the leader in offering a variety of webservices, and the services include software, platform, infrastructure, application, etc. Elastic Computing (EC2) webservice is an example of Amazon Web Services (AWS) to offer infrastructure as a service (IaaS). Amazon offers more than 70 products, can be called services, by the time of the writing this chapter, nearly all of them are cloud-based enterprise solutions including computing, networking, storage, analytics, operations, and security. Apart from AWS, Amazon offers many other webservices internally as well as publicly and follows strategic approach to manage the development of all the webservices at the organization level.

Until Amazon officially launched AWS in 2006, it is an e-commerce company offering online store for retail including books. AWS borrowed lot of learnings from the development of its e-commerce application. Adopting webservices architecture is one of the lessons that AWS imported. Amazon had to take an orthogonal shift in its

development process in 2001 to migrate from its monolithic system to service-based approach. "a senior manager for product management" stated the laborious migration process as "*We went through the code and pulled out functional units that served a single purpose and wrapped those with a webservice interface. We then established a rule, that from now on, they can only talk to each other through their webservice APIs.*" (http://thenewstack.io/led-amazon-microservices-architecture/) (http://thenewstack.io/springone-2gx-conference-managing-migrating-monoliths-mud/).

Not just moving functional units into a webservice, but are made independent from each other. Such independence benefitted organization to delegate ownership of each service to an engineering team and iterate independent from each other. Over time the functional units in webservices turned into features of the application and an engineering team for a feature. An engineering team owns a feature end-to-end starting from drawing a roadmap to deployment and maintenance.

Witnessing the efficiency from service-oriented approach, Amazon later consistently followed the same process across its organization for every application it has been building. Amazon flourished the number of services at scale of hundreds. In addition to adopting the service-oriented approach, Amazon standardized organizational structure within the organization and later many companies adopted the approach. 2-Pizza team and flat engineering teams are some of the popular standards practiced in Amazon and well known in the industry.

Walmart

Until 2012, the Walmart's Global e-commerce system was a monolithic application, deployed once every cycle, nearly 2 months period. Such a long development and release cycle accounts for an unsustainable rate of innovation in comparison with the fast-moving technological advancements. Learnt hard way from the incompetent development and maintenance process, Walmart chartered resources to renovate their software system to support cutting-edge technologies and help their business grow. Such initiative evolved into a massive "Pangeae" project that transforms nearly every functionality in their system into microservices [http://www.oneops.com/ 01/21/2017]. Through the efforts from the Pangaea paved the way, Walmart's e-commerce site was rebuilt in service-oriented architecture. The objective of the Pangaea is to prepare for e-commerce business for 2020, with 4 billion people connected, 25+million apps available, and 5.200 GB of data for each person on Earth.

Further, the Pangaea project is stimulated by acquiring "OneOps" to enable engineering teams an agile, cost-effective, flexible application life cycle management solution in a cloud. OneOps is a cloud management and application life cycle management platform that developers offer to both develop and launch new products faster, and to easily maintain them throughout their entire life cycle. It was a key to enable the project Pangaea, giving the required autonomy for developers and teams allowing them to independently do releases and updates with a minimum amount of effort and little dependency from Ops. Through OneOps, Walmart optimized the process to deploy and maintain microservices in an infrastructure. Walmart scaled the process and now on a typical day it accomplishes over 1,000 deployments, exe-

cuted on-demand by development teams, each taking only minutes on average to complete.

7 Final Words

As webservices are becoming de facto adaptation in nearly all organizations, there has been little research put into standardizing the development process for webservices. An optimized process increases the benefits to reap from the webservices approach, and this chapter attempts to establish such process. Webservices development process borrows many concepts from agile software development process and injects what is needed in specific to webservices. Not just challenges in webservices, but the new advancements in webservices should be addressed effectively in the development process and some of them are discussed in the chapter as well.

References

1. Web Services Architecture, W3C Working Group Note 11 February 2004 https://www.w3.org/TR/ws-arch/#id2260892
2. http://www.ibmsystemsmag.com/Blogs/IT-Trendz/September-2015/Microservices-and-the-Development-Lifecycle/
3. https://www.youtube.com/watch?v=SPGCdziXlHU
4. http://cdn.oreillystatic.com/en/assets/1/event/164/The%20full%20life-cycle%20of%20a%20microservice_%20How%20to%20realize%20a%20fault-tolerant%20and%20reliable%20architecture%20and%20deliver%20it%20as%20a%20Docker%20container%20or%20in%20a%20cloud%20environment%20Presentation.pdf
5. http://3gamma.com/insights/architecting-speed-agile-innovators-accelerate-growth-microservices/
6. https://opencredo.com/versioning-a-microservice-system-with-git/
7. http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/microservices.html
8. http://ceur-ws.org/Vol-243/SMR2-2007-paper7.pdf
9. https://hbr.org/2014/07/speed-up-your-product-development-without-losing-control
10. http://www.slideshare.net/StefanIanta/googleuberservices-servica
11. http://agileconsortium.pbworks.com/w/page/f/XR7+mstriebeck-ShtAddingProcess.pdf
12. http://evelynrodriguez.typepad.com/crossroads_dispatches/files/GoogleProductDevProcess.pdf
13. https://www.oasis-open.org/committees/download.php/13420/fwsi-im-1.0-guidlines-doc-wd-publicReviewDraft.htm#_Toc105485430