University Halle-Wittenberg
Institute of Computer Science

Proceedings of the PhD Symposium at the
1st European Conference on Service-Oriented and
Cloud Computing

Wolf Zimmermann (Editor)

Technical Report 2012/03
Contents

Towards Performance Evaluation in Volunteer Clouds
Abdulelah Alwabel, Supervisors: Robert Walters and Gary Wills 5

On-Demand Service Quality Adjustment using Cloud Computing Techniques
Marvin Ferber, Supervisor: Thomas Rauber 11

Automatic Reliability Management in SOA-based critical systems
Senol Arikan, Supervisor: Paul Müller 17

Termination Analysis of Service-oriented Systems
Mandy Weißbach, Supervisor: Wolf Zimmermann 23

Towards a Model-based Hybrid Service Composition for Dynamic Environments
Aitor Murguzur, Supervisor: Goiuria Sagardui 29

Toward Generic Dependency Management for Evolution Support of Inter-Domain Service-Oriented Applications
Aneta Kabzeva, Supervisor: Paul Müller 35
Preface

The goal of the PhD-Symposium is to provide a forum for PhD students to present and to discuss their work with senior scientists and other PhD students working on related topics. As for the main conference, the topics focus on all aspects of Web Services, Service Oriented Architectures, and related fields. In contrast to the main conference, this work is usually unfinished or has just been started in the PhD projects. The programme committee carefully selected six contributions. Each submission was reviewed by at least two PC-members. In addition to the precise description of the problem to be solved, preliminary results, and first ideas for solving the main problem, the contributions also include a workplan. All these issues are discussed at the symposium with selected senior scientist and the PhD students.

We are grateful to the conference organizer Gianluigi Zavatarro for his organizational support. We thank the international programme committee consisting of

Antonio Brogi (University of Pisa, Italy)
Scharam Dustar (Vienna University of Technology)
Birgitta König-Ries (University of Jena, Germany)
Kung-Kiu Lau (University of Manchester, U.K.)
Cesare Pautasso (University of Lugano, Switzerland)
George Angelos Papadopoulus (University of Cyprus, Cyprus)

for their work. Finally we would like to thank all authors - no matter whether their contribution has been accepted.

Wolf Zimmermann
PhD Symposium Chair

Halle, September 13, 2012
Information about the Author(s)

Wolf Zimmermann (Editor)
Institut für Informatik
Universität Halle
06099 Halle, Germany
E-Mail: wolf.zimmermann@informatik.uni-halle.de
WWW: http://swt.informatik.uni-halle.de/mitarbeiter/zimmermann/
Towards Performance Evaluation in Volunteer Clouds

Abdulelah Alwabel, Robert Walters and Gary Wills
Electronics and Computer Science School,
University of Southampton

{aalal0,rjw1,gbw}@ecs.soton.ac.uk

Abstract. Cloud computing has emerged as a new paradigm that promises to reduce costs of IT by allowing customers to harness computing resources and pay for their usage only. However, cloud computing can still be costly for some projects such as scientific projects. Therefore, volunteer cloud model appears with a goal to provide cloud services at a little cost, if not free. Volunteer clouds aim at providing cloud capabilities out of non-dedicated resources such as non-dedicated PCs based on the cloud business model. However, volunteer clouds present numerous challenges that need to be tackled before it can be seen as a viable solution. The performance of services provided by volunteer clouds is a major issue in this context. This report presents our research problem, motivation and work progress in a PhD research.

Keywords: cloud, volunteer cloud computing, cloud architecture

1 Introduction

Cloud computing represents a shift away from computing being purchased as a product to be a service delivered over the Internet to customers. Economic benefits are the key role behind the appearance of cloud computing [1]. The Cloud transforms IT assets from being capital expenditure to be operational expenditure. Traditionally, small and medium enterprises obtain IT infrastructure by purchasing it. In the cloud, using a server for five hours costs the same as using five servers for an hour [2]. However, the cost of consuming services can be an obstacle against moving toward the cloud. For example, some organisations cannot afford the charge of cloud services in the long term. Therefore a new type of cloud has emerged recently to overcome this limitation. The new type provides clouds’ capabilities based on non-dedicated resources. The new type can be called volunteer or non-dedicated clouds.

Volunteer cloud computing (VCC) can be an alternative choice to the current version of clouds which are offered by commercial companies such as Google and Amazon (to be called commercial clouds). VCCs aim at providing cloud capabilities at no or low costs by harnessing idle resources that are contributed by the public. However, VCC is in its infancy level and suffers from some issues that need attention before it can become a viable solution. This report addresses the evaluation of performance of
VCC service as being a research challenge. In addition, our motivation to participate in this context is discussed. Furthermore, the paper presents a brief background about volunteer clouds and related works. We present architecture for volunteer clouds that can help in solving some issues in volunteer clouds. Our conclusion and future work are presented at the end of this report.

1.1 Research Problem

The services provided by volunteer clouds is expected to be low [3] compared to that provided by commercial clouds due to the nature of the underline infrastructure. This requires a way to evaluate VCC performance in order to be able to enhance it in the future since it has been shown that performance is one of the major concerns in the cloud. However, the literature shows very little work has been conducted in this area. In our research, the intention is to find a technique to measure performance in VCCs. The contribution that we hope to produce by the end of this research as follows:

- Define performance metrics in volunteer clouds: The metrics can be gathered from related computing models such as cloud computing and Grid computing. For example, some performance metrics: response time, resource utilisation and scalability. However, the nature of the used resources to form a volunteer cloud may require new metrics.
- Evaluation of performance provided in volunteer clouds is quite vital. This can be implemented in a tool. The research can lead to present performance benchmarks in volunteer clouds.
- We will try to find a way to predict the performance for each task in a particular volunteer cloud. We will conduct a study to examine whether this tool can apply to commercial clouds.

1.2 Motivation

We are motivated by the fact that building a cloud out of non-dedicated resources can serve the research community in producing a better quality of research. Indeed this allows them to benefit from the cloud services with little or no cost. Furthermore, volunteer clouds can help in preserving and reducing gas emission because they can be formed without building new data centres which have a negative impact on the environment. In addition, volunteer clouds utilise idle resources which means they benefit from those idle resources, provided that approximately 80% of resources in organisations remains idle most of the time according to [3]. Finally, the rapid growth of devices connected to the Internet can constitute a viable way for cloud service providers to exploit them in case their data centres reach their limits.
2 Background

Volunteer cloud computing is based on merging two computing concepts: cloud computing and volunteer computing. Cloud computing is a new computing model that offers shared resources to be accessed online on a pay-as-you-go basis. Volunteer computing is a term which means offering computing resources, such as processing power, to be used by others on a voluntary basis [5]. Volunteer clouds, however, are not limited to resources that are denoted but rather it is more generic to involve any non-dedicated resources used to from a cloud. For example, an organisation may wish to use their local infrastructure to build a cloud.

Nebula [6] is a project aiming to exploit distributed resources in order to create a volunteer cloud which offers services free of charge. Weissman et al. evaluate the performance of Nebula vs. commercial clouds in dealing with highly distributed data-applications [7]. The paper starts by presenting the architecture of Nebula. The model consists of a master node, data nodes, execution nodes and a database. The master node forms the interface between Nebula and its clients and is responsible for managing and assigning tasks to nodes. Data nodes store the data that will be processed by execution nodes. Execution nodes process tasks from the master node and retrieve data from data nodes. The database maintains a list containing information about the master node, execution nodes and data nodes. The list is updated frequently in order to add new or remove unavailable nodes. Experiments were conducted to evaluate the performance of both Nebula-like (a prototype on Planet-Lab [8] simulates Nebula) and cloud-like (a prototype simulates commercial clouds such as Amazon EC2). They used a distributed blog analysis application for their experiment. Overall, the results show that Nebula was better in terms of performance, even given node failure in Nebula, which is highly likely to happen in reality.

Cloud@home is a project representing the @home philosophy in cloud computing [3]. The goal of Cloud@home is to form a new model of cloud computing contributed by individual users. That means using clouds’ clients to participate in building clouds. However, some challenges remain to be solved. The first is that it requires a managing mechanism for services and resources. The management phase involves Quality of Service and Service Level Agreements (SLAs) for the provided services. Secondly, an interface is required between the participants and the underlying resources in the cloud. Furthermore, security is a major issue in cloud computing and it has more impact in volunteer clouds since it must prevent local access. In addition, volunteer clouds require means to interact with other clouds for data migration or to gain extra computing resources. Finally, it is necessary to implement a resource replication mechanism in order to maintain an acceptable level of reliability. The architecture of Cloud@home consists of frontend, virtual and physical layers [9]. The frontend layer represents the interface between clients and Cloud@home. The virtual layer is responsible for providing the frontend layer with virtual machines for execution and storage services by virtualised the heterogeneous physical resources. The physical layer is the group of available resources volunteered by contributors.
3 The Architecture

This section proposes our architecture for VCC. The architecture can be employed in order to overcome issues in VCCs. The most abstract level of the architecture, Fig. 1, divides VCC into three layers in a way similar to cloud@home: (i) a service layer; (ii) a middleware layer; and (iii) a physical layer. We argue that the cloud@home architecture is not detailed enough to cope with some issues in volunteer clouds, performance issues for example.

![Fig. 1. A proposed Architecture](image)

3.1 Service Layer

The service layer is a layer providing services via an interface to customers based on SOA approach. The business model in VCC is similar to that of commercial clouds, which aim to provide metered computing power as a customer desires. VCC’s contributors volunteer their resources to form a VCC for a certain time, and they may be services consumers at the same time if they wish.

![Fig. 2. Middleware Layer](image)

3.2 Middleware Layer

This layer represents the core structure of the VCC architecture. The aim of the layer is to provide resources to the service layer as they would be provided by a commercial cloud. The layer, shown in Fig. 2, consists of task management and QoS management. Task management works with tasks received from the service layer. It involves task scheduler, load balancing and self-automation. The task scheduler organises tasks
coming from the service layer by passing them to suitable resources. Resources are offered by the resource manager in the physical layer. The load balancing ensures that the load is distributed appropriately, thus minimizing the required time to process a task. Self-automation helps to provide the rapid elasticity in VCCs. It allows users to scale services up or down according to their needs. Quality of service management ensures that a minimum quality level is maintained. The performance monitor in QoS management ensures that the performance of each task is maintained at an acceptable level which is reported in the service level agreement (SLA) reporting component. Node volatility is quite high in VCCs, so the performance monitor must cooperate with the resource management to find reliable nodes among available resources that suite each task. The fault recovery component can be vital with regards to improving the performance of the overall VCCs. Fault recovery can employ a number of techniques to improve the availability level.

![Diagram of Physical Layer](image)

**Fig. 3. Physical Layer**

### 3.3 Physical Layer

The physical layer contains nodes contributed by the public. However, the contributors can only be from sources that are trustworthy; predefined organisations for example, according to administrator preferences. The physical layer, **Fig. 3**, is responsible of managing resources. Resource manager is responsible for resource aggregation, resource allocation and resource monitoring. Resource aggregation aggregates volunteer nodes from the public. An aggregation mechanism can classify resources according to a number of criteria with the aim of optimising the quality of service. For example, the history of each volunteer node can be useful in terms of recognising which node should be selected by the resource allocator for each task. The resource allocator receives tasks from tasks management and allocates them to the required resources. The allocator can decrease the interruption of services by assigning tasks to nodes with higher reliability. It can choose nodes with low reliability to be replicated nodes. However, this requires a technique to compute reliability for each volunteered node in volunteer clouds. The monitor component observes allocated resources regularly in case any of them becomes unavailable. In this case, the monitor informs the failure recovery in the middleware in order to recover the task from a replicated node.

### 4 Conclusion and Future Work

In conclusion, this report presented volunteer clouds as a new type of cloud computing based on infrastructure that is made out of non-dedicated resources. The infra-
The structure of volunteer clouds is made of unreliable nodes that can join or leave the cloud without prior notice. The intention is that this research will lead to present performance benchmarks which require a list of metrics in order to evaluate and predict the performance of volunteer clouds. As a first step in this research, we proposed a VCC architecture which can help us in solving the research problem. Our future plan involves: Implement the architecture in order to evaluate it. Then compare the architecture with other available architecture used in commercial clouds. We will define a list of performance attributes for VCCs and implement them in a VCC evaluation system. Finally, based on our results, we will design an algorithm to predict the performance of each task in volunteer clouds.

References

On-Demand Service Quality Adjustment using Cloud Computing Techniques

Marvin Ferber and Thomas Rauber
(PhD Student) (Supervisor)
Department of Computer Science
University of Bayreuth, Germany
email: {marvin.ferber,rauber}@uni-bayreuth.de

Abstract. The provision of Software-as-a-Service has emerged introducing techniques like utility computing and pay-per-use billing for software products. This dissertation work presents a technique how software modules can be exposed as adjustable Web services using cloud technology. The adjustment of the Web service quality at runtime is one of the major goals. Therefore, the behavior of the software module of interest is assessed using runtime inspection. This is done to provide a measure that captures the non-functional attributes like the performance of such services according to their cost. Public Infrastructure-as-a-Service providers are used to provide the virtual compute resources to host the services. On side of the service consumer, a strategy is developed to adjust the Web service quality based on e.g. the available network performance. A novel Web service interface is introduced, that allows for such an on demand Quality of Service (QoS) adjustment. Furthermore, different aspects like modeling and performance of such Web services are discussed.

1 Introduction

In the past years, the use of Software-as-a-Service (SaaS) has moved into the focus of public interest. Advantages like ubiquitous availability and pay-per-use billing are often named [1]. However, prior to usage, such software services have to be provided. Beyond traditional (Web) service enablement approaches, SaaS includes the essential modeling of a service life cycle for billing. This also implies the use of suitable encryption and authentication mechanisms. Furthermore, the advent of utility computing using Infrastructure-as-a-Service (IaaS) providers enables the adjustment of the quality of software services on demand.

Adjustable software services can be used in a variety of use cases. Innovative program modules may be offered as a remote software service on a pay-per-use model in order to use the functionality in different client programs. This implements an implicit licensing and prevents misuse of software products in contrast to physically distributing a software, e.g., on a CD. Furthermore, adjustable software services can also be used to dynamically offload and execute code, e.g., to assist smart phones in computationally intensive tasks. Nonetheless, the use of externalized software modules can also be useful in business processes, e.g.,
Busines Process Execution Language (BPEL), to avoid the reimplementation and to facilitate the reuse of business logic that is already available.

Current approaches for SaaS provision often fall short of providing a measure and suitable adjustment mechanisms in order to adjust the quality of the desired service according to the needs of the service consumer on demand. As an example, it would be beneficial if a user can adjust the processing speed of an image processing service on demand in accordance to the amount of money the user is willing to spend. An overview of the cloud service provisioning scenario is given in Fig. 1. This dissertation work will focus on research how software services can be exposed as adjustable cloud services providing the same service at different quality using IaaS providers such as Amazon EC2\footnote{http://aws.amazon.com/ec2/}.

The article is structured as follows. In Section 2, the idea of this work is presented along with research challenges that need to be solved. Section 3 then presents a plan and methods how to solve the named challenges. Section 4 discusses related work and Section 5 shows preliminary results.

2 The Idea and Research Challenges

The idea of the herein described approach is to create a dedicated service instance of a specified type for each request. To achieve this, virtual compute resources of an IaaS provider are used. This enables service consumers to select a suitable service quality based on the current needs and attributes. Such attributes are e.g. cost, network link speed, or size of the data to process. This is contrary to traditional service selection approaches where a suitable service is selected from a set of available services [2]. Even existing on-demand dynamic service placement strategies are hard to apply, because a device of the desired quality for placement might not be available for each request [3]. By assigning each request to a dedicated service instance, the burden of load balancing is moved to the infrastructure layer. As a result, a fixed quality of each service invocation can be guaranteed, assuming that all resource instances from the IaaS provider achieve a comparable performance.
One research challenge is the development of a decision model for on-demand service selection on side of the service consumer. We take into account the quality attributes performance, cost, and reliability [4]. Tools are necessary to assess the behavior of exposed services at runtime in order to provide a measure how the service scales according to the underlying virtual machine (vm) attributes. The influence of the parameters CPU utilization, memory consumption, and network traffic size have to be investigated. This is necessary, because vm resources of IaaS providers differ in the number of CPU cores, the CPU core speed, and memory configuration. Fig. 2 illustrates the execution time of a cloud assisted version of the raytracing software Sunflow on different vm instances from the IaaS provider Amazon EC2. Thereby, a higher Elastic Compute Unit (ECU) rating reflects a machine with more CPUs or a higher CPU core speed.

A second research challenge is the development of a platform to host such adjustable Web services on top of an IaaS provider. The intended exclusive assignment of user requests to cloud resources conflicts with common billing behavior of the IaaS providers, which often bill resource usage on a fixed temporal basis, e.g., per hour. This also means that resource allocation and load balancing strategies have to be investigated in order to reuse already running virtual machines to avoid monetary overhead.

3 Methodology

According to the research challenges identified, a requirements analysis is necessary to create a model that includes all aspects for the cloud service selection strategy. The requirements analysis includes multiple aspects:

- Target applications are detected that benefit from a runtime adjustment of the service quality. Such application can be taken from Open Source portals such as Sourceforge[3].

Fig. 2: Execution time of a cloud assisted version of the ray tracing software Sunflow on different virtual machine instances from the Amazon EC2 environment.

---

3 [http://sourceforge.net/](http://sourceforge.net/)
– Parameters are identified that influence the service quality decision process, e.g., network link speed or available money. The measureability of these parameters also has to be investigated.
– Cloud resources that are offered by public IaaS providers need to be assessed.
– Middlewares and technologies have to be investigated regarding their applicability to be used to implement adjustable cloud services.

Based on the requirements identified, an abstract model for cloud assisted applications is created that includes the presence and usage of adjustable cloud services. To create adjustable Web services the runtime parameters like the size of input/output data or the parallel speedup regarding the size of the input data is assessed. This information is necessary to predict the possible cloud resource occupation that a service invocation may trigger. This also includes a suitable accounting of invocations on such a cloud service as well as the provision of cloud resources that contain/run the actual services. In this context, resource allocation strategies need to be investigated that provide a good trade-off between cost, performance, and availability. As it can be very costly to measure the performance of cloud resource allocation strategies on real hardware, simulations are performed instead. Therefore, suitable parameters of real IaaS providers need to be measured in order to get accurate simulation results.

Finally, a middleware concept is proposed that allows the provision and usage of adjustable cloud services in Java. A Web Service Description Language (WSDL) interface encoding representing life cycle attributes such as construction, destruction, or a service instance reference is developed. Afterwards, existing Web service standards (e.g., Simple Object Access Protocol (SOAP)) and implementations (e.g., Apache Axis) are investigated regarding their ability to handle this extended WSDL interface. Furthermore, it is investigated how services of different quality can be instantiated on request of a user. Thereby, an automatic forwarding of requests is compared to an explicit usage of a broker/registry like in Java Remote Method Invocation (RMI).

The feasibility of the approach is shown by prototypical implementations of cloud-assisted application from the image processing domain that used the middleware developed. Existing Open source software is transformed manually using refactoring tools like Eclipse IDE. An additional runtime analysis is performed to determine the behavior of the software holistically. Therefore, different methods for runtime inspection are investigated. The Java debugger, Bytecode instrumentation techniques, or Aspect-oriented programming can be used. It is required that a runtime investigation does not influence the program behavior significantly, because the performance of the software should be assessed.

4 Discussion

Communication middlewares like RMI or RESTful Web services seem more dedicated to the field of distributed processing than SOAP Web services. However, all those technologies are comparable regarding their functionality. The significant point to use Web service technology is that SOAP Web services are highly
standardized and many useful extensions are available already, e.g., WS-Policy, or WS-Security.

Other middlewares for on-demand code offloading from mobile devices such as MARS [5] do not consider the allocation of remote resources as offload target. Instead, their focus is more on energy saving. As a result, these middlewares are often restricted to the usage of WLAN and are hard to apply to publicly available cellular networks. In contrast, our approach is built on public infrastructures and can thus be applied easily.

There exist various approaches to select Web services based on QoS constraints. Some of these approaches even negotiate an SLA at runtime [2]. However, none of these approaches is able to really create a service of a desired quality on demand. Instead, a service is selected from a pool of available instances.

5 Preliminary Results and Outlook

As a first step, it has already been shown that SOAP Web services can be used to express stateful (remote) object behavior within the WSDL specification. Therefore, a special dialect of the WSDL has been introduced as Remote Object over Web service (ROWS) [6]. In another work, this ROWS interface has been used successfully to provide cloud services of different quality for the purpose of remote processing on resource constrained mobile devices [7]. As an example, the Open Source raytracing software Sunflow has been transformed successfully into a distributed version that uses Web service technology (ROWS) as communication middleware. Thereby, a middleware implementation using a dedicated broker entity has been used to allocate resources and perform authentication.

A tool suite has already been developed that allows the runtime inspection of Java programs based on bytecode instrumentation. One tool creates package,
class and call dependency graphs. A second tool enables the temporal monitoring of access to class instances from multiple threads. Finally, the size of data structures can be assessed at runtime by a third tool. Using this tool suite, the behavior of a Java interface can be investigated in order to measure and visualize the necessary attributes like parallel speedup, input/output data size, or memory consumption for a particular Web service invocation. This tool suite has been evaluated for the purpose of cloud service enablement using different software. Exemplary visualizations for the software Face Detection in Color Images\footnote{http://sourceforge.net/projects/facedetectionin/} are given in Fig. 3. An extract of a call graph visualization is shown on the left. The Interface under investigation is marked red. In the middle, the corresponding temporal sequence of method invocations on an object of this type are shown. Self invocations of public methods are also visualized. On the right, a corresponding log file is shown, which also includes the size of input and output parameters/return values that were measured at runtime using serialization.

Using the runtime inspection in addition to static software analysis techniques, we have already been able to transform a variety of Open Source Java software into cloud services. We have investigated the influence of the vm CPU configuration (single core speed and number of cores) to select a suitable cloud resource. Furthermore, we investigated how the network quality between the client and server may influence the performance of cloud service invocations. The next step is to investigate how different configurations of main memory can influence the behavior and cost of cloud services to complete the model description.

References

Automatic Reliability Management in SOA-based critical systems

Senol Arikan
Integrated Communication Systems Group
University of Kaiserslautern, Germany
E-Mail: arikan@informatik.uni-kl.de
Supervised by Prof. Dr. Paul Mueller
http://www.icsy.de

Abstract. A well-known concept for the design and development of distributed software systems is service-orientation. In SOA, an interacting group of autonomous services realize a dynamic adaptive heterogeneous distributed system. Because of its flexibility, SOA allows an easy adaptation of new business requirements. This also makes the service-orientation idea a suitable concept for development of critical software systems. Reliability is a central parameter for developing critical software systems. SOA brings some additional requirements to the usual reliability models currently being used for standard software solutions. In order to fulfill all requirements and guarantee a certain degree of reliability, a generic reliability management model is needed for SOA based software systems. This article defines research challenges in this area and gives an approach to solve this problem.

Keywords: Distributed systems, SOA, critical systems, reliability, fault tolerance

1 Introduction

Service Oriented Architecture (SOA) provides a dynamic and adaptive solution for building distributed systems. Large software systems are built using loosely coupled, autonomous services that dynamically bind and discover each other through standard protocols. It allows for easy integration of old systems and rapid adaptation of new requirements. These characteristics make service orientation a suitable concept for development of critical IT landscapes especially for business critical systems.

A critical system is one where failures may have grave consequences, e.g. physical harm, environmental harm, high financial loses. Well-known examples of critical systems are medical devices, aircraft control, nuclear systems, and financial software systems like online banking [1]. It is essential to design and make those systems as dependable as possible before their complexity becomes too overwhelming and handling failures becomes too difficult [2]. One of the most important conditions for developing a dependable software system is reliability.
IEEE 610.12-1990 [5] defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." The main purpose behind focusing on reliability is to guarantee that the resources managed and used by the system are under control. It must be guaranteed that a user can complete its task with a certain probability when it is invoked. Software reliability management is defined in IEEE 982.1-1988 [6] as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and use of measurements to maximize reliability in light of project constraints such as resources, schedule, and performance". From these definitions the following classification of methods to develop reliable software systems can be derived: fault prevention, fault tolerance, fault removal, and fault forecasting. Reliability measurement plays a particularly important role in these methods, which are described in the next sections.

A software reliability model is comprised of certain assumptions about the system, relevant factors about the software, and mathematical functions relating the reliability to the factors. The most suitable models for reliability of SOAs are the ones based on architecture. The main reason is that they focused on the components and on the control and data flow between components. There are three distinct approaches in architecture-based reliability modelling: state-based models, additive models, and path-based models [4] [7]. State-based models analytically estimate reliability by using the control flow graph of a software architecture. Path-based models compute all possible execution paths, then calculate the reliability of each path by multiplying the reliabilities of the components along that path. The system reliability is the average path reliability over all paths. Finally, additive models describe each component's reliability and the system reliability as Nonhomogeneous Poisson Processes (NHPP)[4].

SOA brings new challenges to the usual reliability models currently in use:

- Software systems are commonly developed and maintained by one entity with full control of its development. In SOA, the entire software system consists of an interacting group of autonomous services. These services are developed and provided by different (geographically distributed) stakeholders and they can be executed separately or together in a workflow. Each service provider offers varying degrees of reliability, and there is no guarantee that a service will be available and operating correctly.
- All major elements of the architecture are loosely coupled, which means that they have to be evaluated separately (with potential lack of information). Reliability should be guaranteed for the basic service, for the data flow, for composition of services and for the complete workflow.
- Service-oriented architectures are by nature dynamic, with service publication, registration, discovery, binding, and composition occurring while the system runs. During runtime new services can be added, existing services can be removed, and running services can fail. This dynamic character of SOA requires a reliability model which reacts at runtime to system changes to ensure accessibility of services.
Another point is the statelessness of services. In SOA, the services do not store any state information about a task. It means all data will be lost if a task gets interrupted. This problem must be also considered by a reliability model for SOA.

The above mentioned characteristics challenge new reliability models and solutions. The rest of this paper will discuss existing reliability solutions, define research challenges and describe a proposed solution for the defined problems.

## 2 Related Works

SORM, Service-Oriented Software Reliability Model [10], is one of the earliest architecture-based reliability model for SOA. It consists of two stages: group testing to evaluate the reliability of atomic services, followed by the evaluation of composite services through the analysis of components and their relationships. In this work, Tsai et al. use a group testing technique [11], from the medical field to detect faults. Before further discussion, three assumptions about reliability in this model should be discuss. First, assignment operations never generate new failures. Second, a condition fails when any data associated with it fails. Finally, there is no cyclic dependency among ACDATE [10] entities.

In [12] a new service, ReServE, is proposed that will transparently recover the state of a business process, ensuring that it is consistently perceived by client and services. The main component of the ReServE-architecture, the Recovery Management Unit - RMU, records all communication (requests and responses) between client and services. For each service there is a Service Proxy Unit (SPU) that monitors the service and detects the failure. When a service fails, its SPU can initiate the rollback-recovery process.

A unified reliability modelling framework is defined in [3]. It provides a hierarchical reliability model. The reliability of simple services is addressed by considering data reliability and service pools as backup alternatives. Discrete Time Markov Chains (DTMCs) are used for analysing service composition reliability.

Using information published with each service, [13] presents an architecture-based reliability prediction model for SOAs. In this work, reliability is viewed as a measure of a services ability to successfully carry out its task when invoked. The final goal is to have an automatic and compositional reliability prediction method.

Every solution discussed above has its own focus and meets a part of SOA requirements. For example, according to the definition of Erl, statelessness is a principle of service design [15]. If a service request is interrupted, all execution data will be lost. A restart of the service request from the beginning will waste resources and produce unnecessary cost. The idea of Danilecki et al. [12] provides a solution to recover state of a service request but it doesn’t solve whole reliability problem. The solution of Tsai et al. [10] focus on the proper detection of failures. They assume the existence of a set of equivalent services that could be used in a voting system, to ensure identifying of unavailable services. Fault forecasting is
not addressed in this solution. As a countermove, the proposed model in [13] can be focused on fault forecasting. Each related work presents a partial solution for reliability. In order to get a complete solution, first it is required to analyse characteristics of SOA and define corresponding requirements. After that, it should be discussed which existing solutions can be used to fulfill the defined requirements and which solutions should be newly developed. The next section discusses research challenges and describes a solution approach.

3 Discussion and Research Approach

In order to optimize the reliability of a system and to satisfy runtime requirements such as use of resources, performance, etc., a reliability management model is needed. The required model must be able to define failure-critical system components and decide the most suitable reliability technique. The redundancies for failure-critical (or failed) services must be deployed and configured automatically. This step alone implies some important questions which must be considered. The configuration and generation of redundancies and than the deployment and integration of them into running system (including service discovery, service binding, service execution) are most important issues. The focus of this PhD will be dealing with these issues to develop an automatic reliability management in SOA. The rest of this section presents the first architectural draft (see Fig. 1) of the proposed reliability management concept. It provides a general overview and describes some future contributions.

In Service-Oriented Architectures, the entire system can be separated into three layers: business layer, services layer and hardware layer[15]. The reliability of the target system must be considered on all these layers. Reliability can be guaranteed in three steps: fault detection, fault forecasting and fault elimination[9]. First the whole system must be monitored to detect failed system components (fault detection). Second, the desired reliability model must validate the reliability of all system components to identify possible faults before system failure (fault forecasting). Then the redundancies must be defined and automatically switched on when required (fault elimination).

In Fig. 1, the proposed solution architecture consists of different modules which map the steps defined above. The Module Monitoring component monitors all system components and inputs. It identifies failed components and informs the Management Module. This module also analyses inputs to identify malicious attacks (such as DoS attacks) on the system. The Module Monitoring component collects some data about system components to support runtime learning process of reliability model (for example how many request per second can be executed - capacity of system components). The Module Reliability Validator calculates and analyses the reliability of system components. The fault forecasting is essential for supporting fail-safe critical systems. In the context of this work, it will be focused on this step. The fault forecasting can be reached through an successful reliability analysis of entire system (workflow). There are some reliability analysis techniques, which can be also used for SOA-based systems. The most commonly
used of them are: Fault Tree Analysis (FTA), Reliability Block Diagrams (RBD), Petrie Nets and Markov Analysis [8]. An important part of this PhD work will be choosing the most suitable technique to simulate the reliability of complex SOA systems. The next step must be development of a reliability model with using of the chosen analysis technique. On the basis of the results of performed reliability analysis, the reliability of the whole system (before and after redundancy) and system components can be calculated and the services with the risk of failures can be identified. Thus service users can be supported upon reaching desired reliability.

The *Module Redundancy Installer* generates and deploys redundancies. It configures the redundancies and integrates them into the system. The *Module Request Splitter* distributes requests on the redundancies (through using some load balancing solutions [14]), if a system component achieves its maximum performance or redirects requests, if a system component fails. The *Module Reliability Management* collects data from other modules and makes runtime decisions for an appropriate reliability strategy, redundancy strategies, consensus strategies etc. It also manages the configuration and integration of the redundancies into the entire system. If the system is under outside attacks, the *Management Module* can abort inputs to avoid undesired system failures. Additionally, it notifies clients about modifications in the system.

4 Conclusion

In order to build dependable critical systems, we must consider reliability. Reliability ensures system users a performance continuity despite failures of weak system components. Dynamic and adaptive SOA systems entail some additional requirements to the common reliability solutions. To meet these requirements and to ensure reliability of SOA based software systems, a generic reliability
management model is needed. The contribution of this PhD will be developing an efficient solution approaches to optimize the reliability of SOA. This paper analyses several existing proposals from other researchers in this area. It defines some important requirements for SOA reliability, starts to design first architectural draft. This paper provides an overview of what has been achieved so far and shares with you some thoughts for the future work.

References

Termination Analysis of Service-oriented Systems

Mandy Weißbach
Institute of Computer Science
University of Halle
06120 Halle (Saale), Germany
mandy.weissbach@informatik.uni-halle.de
Supervisor: Wolf Zimmermann

Abstract. Today there exist a wide range of methods to ensure the right behavior of Service-oriented Systems, e.g., proving safety or liveness properties using Petri Nets. Most of these methods assume termination [1]. The subject of this project is to find an appropriate approach to verify termination of existing Service-oriented Systems. This work focuses on existing methods to prove termination of these Systems which can include internal and external concurrency (parallelism), synchronous and asynchronous method calls, recursion and for my assumption recursive callbacks.

1 Introduction

Services in Service-oriented Systems (SOSs) are self-contained and they can be used over interfaces realizing the blackbox view. The reason for the popularity of SOS are the various promised properties (e.g., scalability, reliability and ubiquitous access [2]). SOSs have to have the right behavior to satisfy these properties. If within one component or service in a system an error occurs or even worse, if an error occurs because of the composition of two or more services, promised properties of a SOS can not be guaranteed. For that reason many analysis tools exist. Wendy is a Petri Net based tool to synthesize partners for services [3], Palladio provides methods to construct software according to a set of predefined quality of service attributes for Component-based Systems [4] and another tool is VeriCap, a protocol conformance checking tool [5].

Some tools are based on Petri Nets. Onto fairness assumptions, these tools implicitly assume termination [1]. For proving termination of programs, many tools are available. For example, COSTA is a cost and termination tool for Java Byte code [6]. All termination analysis tools verify the termination of software pieces written with a certain programming language and assume that the whole program is available. Termination analysis tools for SOSs, to the best of our knowledge, do not exist.

To fill this gap, the idea is to use existing methods for termination analysis to examine existing SOSs towards termination. Loops, loops associated with internal and external concurrency, synchronous (the caller waits until the message
is passed to the callee) and asynchronous (the caller does not wait until the message is passed to the callee) method calls and recursion, including recursive callbacks, should be considered. Our goal is to integrate the desired information into the services’ interface description while keeping the blackbox view. The termination analysis should work fully automatically. In case of concurrency, our notion of termination includes deadlocks and livelocks.

The upcoming paper shows the goals, identifies the problems and the main idea in Section 2. In Section 3 two examples are used to show and discuss the results. The related work is discussed in Section 4. At the end, Section 5 gives a short conclusion and an outlook.

2 Problem Statement and Examples

It needs to be examined whether all possible paths within a service can terminate and whether all possible paths within a SOS can terminate. Services are allowed to communicate synchronously or asynchronously. There are no restrictions on recursion (including recursive callbacks) and concurrency such as e.g., bounds on recursion depth or the degree of concurrency. Thus, the following points need to be considered:

1. determining areas of improvement by means of examples [1],
2. termination analysis (TA) of SOSs with no asynchronous calls and no recursive callbacks [7],
3. TA of SOSs with recursion and recursive callbacks (this paper),
4. TA of SOSs with only concurrency including asynchronous calls,
5. TA of SOSs including 3 and 4,
6. implementation of an automatic verification tool.

Literature research has shown that the common consideration of recursion and concurrency in termination analysis of SOSs are missing. In [1] we pointed out the problems that occur if analysis tools based on Petri Nets assume loop termination (fairness assumption). This can lead with an existing non-terminating loop to a livelock situation and if this service was called synchronously by another service it causes a deadlock of the whole system.

The next step is to consider recursion and recursive callbacks. The problem w.r.t. termination is demonstrated by the following two examples.

The Web Services of Fig. 1(a) realize the divide and conquer sorting algorithm Mergesort. Web Service WS Split splits a list into equally sized sublists which are sorted by the Web Service WS Mergesort. The Web Service WS Mergesort provides an interface with a function sort. It uses Web Service WS Split to obtain two sorted sublists. WS Split provides a function split. Web Service WS Merge merges two sorted lists to a sorted list. For this purpose, the merge function is offered by the provided interface of WS Mergesort.

Fig. 1(b) shows an Expression Evaluation system. Both Web Services provide the evaluation functionality over eval in their provided interface. Also, they require the function eval. The Web Service WS Mul evaluates an expression. If
the expression is a multiplication, the operands of the expression are evaluated using WS Add and the results are multiplied. Thereby, the operand expressions are passed to the Web Service WS Add using eval. If an addition expression is passed to WS Mul, then it is delegated to WS Add for evaluation. The Web Service WS Add works analogously: a sum is evaluated using the evaluation service WS Mul for the operands, a multiplication is delegated to WS Mul.

Both examples include recursive callbacks. In Fig. 1(a), the Web Service WS Sort calls the Web Service WS Split, and Web Service WS Split calls the Web Service WS Sort. In Fig. 1(b), Web Service WS Mul calls the Web Service WS Add and vice versa.

3 Results

In this section a systematic approach to gain the stated goals in section 2 is presented. We introduced the size-change principle with termination functions to state termination [1], [7]. These functions are provided over the interface description. Point 1 and 2 of the stated goals are completed. Next, the termination of SOSs including recursion and recursive callbacks need to be examined. For that issue recurrence equations specifying an abstract termination behavior are introduced. Figure 2 shows the workflow to generate for a given system of services, a system of recurrences which can be used to prove termination. First, each Web Service is analyzed on its own. Known compiler methods [8] can be used to compute the recurrences and for that the termination function. In the next step, the combination of all the termination functions to a closed system
of recurrences can be computed. Applying mathematical methods can solve the recurrences [9]. A solution of the system of recurrences proves termination. If no solution can be found, termination cannot be proven. The following recurrences are derived for the example in Fig. 1(a):

\[ \text{Mergesort.} \tau_\text{sort} (n) \leq 1 + \text{Split.} \tau_\text{split} (n) + \text{Merge.} \tau_\text{merge} (\sigma_\text{split} (n)), \]
\[ \text{where } \sigma_\text{split} (n) \leq \left( \left\lfloor \frac{n}{2} \right\rfloor, \left\lceil \frac{n}{2} \right\rceil \right) \]
\[ \text{is the pair of lists with the size of the splitted list.} \]

\[ \text{Mergesort.} \tau_\text{sort} (1) \leq 1 \text{ for Web Service } \text{WS Mergesort} \]
\[ \text{Split.} \tau_\text{split} (n) \leq 1 + \text{Mergesort.} \tau_\text{sort} (\left\lceil \frac{n}{2} \right\rceil) + \text{Mergesort.} \tau_\text{sort} (\left\lfloor \frac{n}{2} \right\rfloor) \text{ for Web Service } \text{WS Split.} \]
\[ \text{Merge.} \tau_\text{merge} (k, n) = 1 \text{ for Web Service } \text{WS Merge.} \]

Note that the parameters \( n \) and \( k \) are the size of the list and \( \text{split} \) divides the list into two equal-sized halves and sorts them recursively. Following the workflow, the System of Recurrences can be computed and solved. Thus, termination is proven, it can be derived that \( \tau_\text{sort} (n) \leq 3 \times 2^n - 3. \)

Consider the example shown in Fig. 1(b). The following recurrences can be computed:

\[ \text{Add.} \tau_\text{eval} (0) \leq 1, \]
\[ \text{Add.} \tau_\text{eval} (n) \leq 1 + \max (\text{Mul.} \tau_\text{eval} (n), 2 \times \text{Mul.} \tau_\text{eval} (n - 1)) \text{ for } \text{WS Add,} \]
\[ \text{Mul.} \tau_\text{eval} (0) \leq 1, \]
\[ \text{Mul.} \tau_\text{eval} (n) \leq 1 + \max (\text{Add.} \tau_\text{eval} (n), 2 \times \text{Add.} \tau_\text{eval} (n - 1)) \text{ for } \text{WS Mul.} \]

Note that the parameter \( n \) is the size of the list. Here, termination cannot be proven since \( \text{Add.} \tau_\text{eval} (n) \leq 1 + \text{Mul.} \tau_\text{eval} (n) \leq 2 + \text{Add.} \tau_\text{eval} (n). \) This cannot happen in real executions since not both, \( \text{WS Mul} \) and \( \text{WS Add} \) just delegate this expression to be evaluated (either it decomposes the expression for evaluating the operands). If a program considered completely, this situation is detected. The challenge is to investigate generalizations of recurrences that allow to detect such situation without giving up the blackbox view of services.

It has to be mentioned that this approach is conservative. Recursions and recursive callbacks may be computed although they never occur when the program is executed. So, false positives can occur but false negatives can not occur.

### 4 Related Work

One of the few works considering program analysis of SOSs is [10]. They consider response time in terms of some notion of input size. Response time information
is provided by the Web Service interfaces. Their approach generalizes the approach of [11] for the analysis of software complexity of BPEL processes towards response time. It seems that size change functions or recurrences play no role in their approach.

Recurrences associated with termination analyses is done e.g., by Albert et al. [6]. COSTA is a termination analyzer for Java Byte code. It computes a recurrence relation similar to ours. Callbacks or recursive callbacks are not allowed.

Process algebras are widely used to specify and verify service-oriented systems. [12], [13] use a process algebra approaches to design and verify Web Services and their composition. [14] provide a solution to the central issues of orchestration and choreography of Web Services. A contract-based approach is presented by [15]. They also consider the fair termination property to ensure that all services terminate. Termination or termination analysis in our sense is not considered in these works.

Research work concerning termination of business process workflows are using the Petri Net representation. The approach of van der Aalst et al. [16] uses Petri Net properties to show soundness. With the assumption of strong fairness, they state termination with the remark one should not introduce infinite loops.

With these approaches deadlocks in context with lacking resources can be found [1]. To our knowledge, the proof of termination in combination with the occurrence of infinite loops, concurrency, recursion and recursive callbacks in SOSs are not investigated by other researchers except [1].

5 Conclusions

A short overview of the research topic termination analysis of SOSs was given. All known analysis tools concerning safety and liveness properties are based on Petri Nets. They assume termination of every service within a system. It was shown that without concerning termination, real life behavior can not be considered. Termination work also exists but, to my knowledge, nobody else is concerned with termination analysis of SOSs. The termination of loops with the help of size-change function was shown in [1]. In this paper, the termination of recursive callbacks with the help of recurrences was shown. Recurrences can be automatically computed and solved. It has to be mentioned that this is a typical conservative approach; if termination can be proven, then the SOS indeed terminates. However, if termination cannot be proven, the SOS may still terminate.

On Petri Net based analyses assume termination [16]. With our approach this assumption can be justified. Analyses based on Petri Nets would have a sound foundation. Recursive callbacks combined with concurrency, further implementation of a fully automatically tool to test the practical suitability and applicability are still open points to be investigated. Furthermore, it has to be pointed out how the approach can be applied to SOSs keeping the business secret and the blackbox view.
References

Towards a Model-based Hybrid Service Composition for Dynamic Environments

Aitor Murguzur\(^1\) (Student)
Goiuria Sagardui\(^2\) (Supervisor)

\(^1\) IK4-IKERLAN Research Centre, Mondragon, Spain
amurguzur@ikerlan.es
\(^2\) Mondragon Unibertsitatea, Mondragon, Spain
gsagardui@mondragon.edu

1 Introduction

Nowadays, service research has to face the ultra large scale and heterogeneity of the Future Internet challenges, which are orders of magnitude higher than those of today’s service-oriented systems\([1]\). Service proliferation is growing and RESTful services are being used by many Web applications, enabling lightweight service description and invocation through loose coupling. Exposed services are offered to customers as software, infrastructure, and platforms leading to the concept of Everything as a Service, which is the core of cloud computing\([2]\). SOC and cloud computing also have a reciprocal relationship, as Wei et al.\([3]\) highlight, one provides the computing of services (the former), and the other (the latter) provides the services of computing.

Amidst this multitude of available services, an important property of services is their composability. In a general sense, service composition can be seen as a combination of a set of services for achieving a certain purpose\([4]\). The resulting service can be used as a new service offering functionality to a service client. During the last decade, service composition has received a lot of attention in literature surveys\([4–7]\), mainly devoted to conventional SOAP-based Web services. However, in a dynamic heterogeneous environment such as the Internet, Web service composition needs to implement fundamental changes to how software is developed including new requirements such as flexibility, adaptation, and quality assurance.

2 Dynamic Heterogeneous Environments

In dynamic heterogeneous environments characterized by low predictability and constant change, service composition should include further requirements. Heterogeneity comes from the spread of an approach in several domains, where a solution should be applied to different areas, enabling domain independence to facilitate a solution for a broad range of problems. The dynamic nature of changing environments results from three factors: the time of the decision making which determines the phase in which changes can be identified and managed,
the degree of change predictability which refers to how easy it is to deduce in advance changes that occur during the process execution, and the degree of change of the environment which is related to the number of changes that happen in a specific domain. For instance, in a healthcare scenario, patient diagnosis cases are really singular. Although some of their activities may be automated, most of the situations are treated in a particular sense, where knowledge workers often decide the right way to handle exceptional situations based on their training.

In a highly dynamic environment (the upper right-hand quadrant of Figure 1), due to instability, functional properties are required but non-functional characteristics are also necessary to guarantee the best possible quality of service. Therefore, QoS-awareness should be also considered. The environment characterization is aligned with the key challenges posed by the Future Internet vision, specially with adaptability and heterogeneity [1]. In such dynamic environments, loosely specified models should be completed at run-time and should cope with exceptional circumstances. Nevertheless, what is predicted to be the next significant trend in Web service composition considering flexibility (uncertainty), adaptability (constant change), and QoS-awareness properties?

3 Overview of Related Work

The manual implementation of Web service compositions is a costly task, so an automated composition is proposed to regularly make up a set of existing services in order to satisfy some given composition goals, as extracted from the systematic review of the state-of-art. Model-based, Workflow-based, and AI Planning-based service compositions may be considered as alternatives.
Model-driven concepts are commonly used in software composition, abstracting the software development to a higher level [4–6]. Even so, in a highly dynamic environment, automated Web service composition approaches are normally adopted. Workflow techniques aim to create a workflow for later execution. Dustdar et al. [8] divide workflow techniques into two sub groups: (i) static which means that the requester should build an abstract process model before the composition planning starts (binding and service selection at run-time), (ii) dynamic composition that creates process models and selects atomic services automatically at run-time. Workflow-based service composition and process flexibility by underspecification [9] seem to be parallel in the survey proposed by Kapuruge et al. [10]. In a static generation, an abstract model is pre-defined at design-time and the selection of services is done at execution, this seems to be in parallel with late binding and late modeling techniques. In contrast, in a dynamic composition, the process model creation and the selection of services are done automatically at run-time, as late composition denotes. Process flexibility and adaptation are two of the major issues [11].

AI planning techniques have been proposed in order to solve the complexity of the service composition task, employing an algorithm for solving a planning problem starting from an initial state and determining the required actions to reach the final goal. Services, usually semantically annotated, are described, discovered and finally composed to comply the desired goals. Although most of the time the planner gets a solution (known NP-hard problem), replanning capabilities are not considered in most of the approaches. AI planning techniques are widely proposed in the extensive SOAP-based Web service composition approaches [4, 5, 7].

In service composition life-cycle three other phases can be distinguished [4]: service execution, service adaptation and service monitoring. All of them are closely related, mainly centered in adaptation support. Adaptation is becoming the new need of the near Future Internet, therefore, any software solution should include adaptation mechanisms to ensure running executions, for instance employing AOP or DSPL [12, 4].

4 Problem Statement

Although regarding to SOAP-based service composition some open research challenges are observed (e.g., adaptability in late composition approaches or semantics integration in late binding solutions), REST is gaining momentum [13]. As it is remarked, to the authors’ knowledge, only few efforts have gone into the wider task of harmonizing the research efforts of RESTful Web service composition. For instance, recently Zhao et al. [14] propose a formal definition of RESTful Web services and provide a two-stage composition. This approach, together with the existing proposals, do not cope well with the following requirements.

- **Flexibility**: Requirements dealing with uncertainty should only redefine what is necessary and routine (known) in a dynamic environment; the composition should be completed at run-time and should give user guidance
when required. Therefore, service compositions should be flexible. With the dynamic nature in mind, flexibility can be achieved by late composition. Adaptability and QoS integration are two of the key challenges of SOAP-based late composition solutions. Not in vain, in RESTful Web service composition these features are still not well thought-out, so there are a new set of research challenges for the near future [15].

- **Adaptability:** Changes in the dynamic environment are constant, which means that composed services should manage and deal with changes at runtime. The late binding technique has been implemented in several SOAP-based approaches [10]. QoS-awareness and self-management mechanisms have also been considered in these solutions in order to deal with constant change situations. An automated service composition with run-time adaptability support should be provided with the aim of being consistent about uncertain, constantly changing situations. Moreover, replanning capabilities, that are suitable for this dynamic nature, should be considered, but also QoS features should be included to ensure an optimal performance rate.

- **QoS-awareness:** As SOC evolves into a major paradigm covering many application domains and quality-critical applications, QoS should be treated accordingly. Numerous efforts have produced QoS languages and models for general or domain-specific QoS description [1]. QoS is considered mainly in SOAP-based late binding service composition approaches, leaving aside the RESTful paradigm.

- **Semantics integration:** Semantics will take as basics in the near future [4, 1]. RDF and Linked Data will offer a new landscape in which to play towards a fully Semantic Web [16]. Therefore, all services will need to include semantic annotations to facilitate the discovery and composition of Web services. Although some of the SOAP-based matching are still unexplored [17], RESTful Web service matching is a more challenging issue [18]. Semantic RESTful initiatives (e.g., hRESTs+SA-REST3, RESTdesc4, etc.) have been already proposed.

Distilling the results of the state-of-art overview and taking the open problems, our approach intends to combine service composition in dynamic environments with the more specific problem of QoS-aware and adaptable hybrid service composition. Given the very big differences between two styles [19] and the new problems within RESTful service composition from [15], our approach will provide seamless support for both WS-* services and RESTful services.

## 5 Hybrid Service Composition in Dynamic Heterogeneous Environments

Our research plan intends to propose a flexible and adaptive approach that can deal with the dynamic environment’s features. A mixture of Model-driven and

---

3 [http://knoesis.wright.edu/research/srl/projects/hRESTs/](http://knoesis.wright.edu/research/srl/projects/hRESTs/)

4 [http://restdesc.org/](http://restdesc.org/)
planning techniques may be used to achieve the above mentioned features. Service description may include semantics and QoS annotations for WS-* services and RESTful services, so a more extensive research will help to point in the right direction. If services are descriptive in a machine processable way, we will be able to crawl the Web finding new service functionalities. As the Future Internet leads [1], a hybrid architecture (relying on a hub and a spoke management layer and a peer-to-peer system) will bring us the right balance of performance and scalability to the next solutions. For instance, crawling the Web to index services and store them in private registries.

Moreover, model-driven techniques will offer the possibility to define meta-models that will be used as a core element for input model description and composition. The model generated as input for a planner (flexible model) and the final composite service model generated by a planner (adaptive model) will conform to meta-models. This allows us to obtain a flexibility by underspecification deferring the whole service composition to enhancement. Run-time adaptability capabilities, such as exception handling or replanning of a planner, will attempt to solve constant change situations based on an adaptive model and will consider non-functional properties to achieve acceptable performance rates of the execution of a composite service.

6 Research Objectives and Contributions

This work pursues the following research objectives (O) with their expected contributions (C):

- **O1**: Integrate QoS in service descriptions, providing a specification of the integration with an existing approach that does not involve complex specifications.
- **C1**: Formalize a lightweight specification that includes QoS features in a service description language.
- **O2**: Provide a flexible and adaptive hybrid service composition approach able to deal with uncertainty and constant change.
- **C2**: Enable a hybrid service discovery mechanism based on functional/non-functional service matching.
- **C3**: Define and implement an adaptive composition system able to deal with constant change situations based on QoS.
- **C4**: Facilitate the service composition creation for domain-knowledge users, being fully responsive of composition life-cycle.
- **O3**: Assess the usability of the approach by empirically applying it to a specific domain (e.g., energy, healthcare, or manufacturing).
- **C5**: Define a series of examples to check the usefulness of the approach.
- **C6**: Define an experiment to analyze the performance of the approach in the adaptation phase.
References

Abstract. Service-orientation is a widely accepted paradigm for the realization of distributed systems and the number of deployed services is constantly increasing. Through the reuse of services in different intra- and inter-organizational business processes, the number of relations between services and between services and processes is growing even faster. Understanding and tracing these relations is a prerequisite to preserve the consistency of all applications after a modification of any part of the system. While a number of research works exist on dependency modeling for support of individual management processes, a generic model for dependency management through all abstraction levels of SOA is still missing. This PhD proposal aims at defining a generic dependency modeling approach to support the management of inter-domain service-based applications during their whole life-cycle and to provide a basis for consistent evolution.

Keywords: Dependency management, service-orientation, inter-domain applications

1 Introduction and Motivation

Driven from the enterprise needs for automation of business to business communication and Enterprise Application Integration, the service-oriented architecture (SOA) paradigm has become a standard for the realization of distributed software solutions. Based on the core concept of a service as a mechanism for accessing a capability through a standardized interface, SOA defines “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [12]. Services have to be designed to be loosely coupled to minimize dependencies and thus increase the flexibility and reduce the impact of modifications and failures within the system [8].

Although addressing the adaptation requirements of business processes, SOA-based systems increase the complexity of the underlying IT infrastructure. There are three factors causing this complexity. First, the construction of applications
as a composition of loosely coupled services increases the number of software artifacts to be managed [9]. Second, many of these independent services are reused and smoothly combined in many business processes [13], which leads to an increasing number of connections between services. Third, the integration of services across ownership boundaries introduces external resources within the infrastructure needed for its proper functioning, which are controlled by third party owners [7]. Changes conducted by the providers of such external services can cause disruptions of client applications [14].

To efficiently exploit all the advantages provided by SOA, organizations need to learn how to manage the complexity that SOA causes as it grows. The successful usage of services leads to additional requirements. New services are introduced and existing services are modified on the way to adapting business processes to these new requirements. Although services are loosely coupled, modification of a single service can affect numerous processes and vice versa. The research on evolution strategies for service-based systems has become a growing issue [11]. Information on dependencies resulting from service composition and reuse can support the evolution process in multiple ways during the whole system life-cycle.

The evolution of a software system is a continuous process which starts with the initial development and continues through its maintenance until retirement. During the design process, information on dependencies between business activities and the technical services implementing them can help in recognizing the reuse potential of existing artifacts [5].

During maintenance, change management is also reliant on dependency information [14]. Flexibility and adaptability are the main reasons for adopting SOA. Change requests can come both from the processes as a reaction to changing market conditions, or from the IT after detection of failures. In both cases, estimating the impact of change requires tracing the dependencies to and from the affected artifacts.

Performance governance of service-based infrastructures can also profit from the collection and documentation of dependency information. The importance of a service for the reliable operation of a system landscape can be estimated based on the number and type of its dependencies. Such information can support decisions on replication or retirement of artifacts, or restructuring of the infrastructure.

2 Research Challenges

In general there are three issues that have to be addressed for dependency management in loosely coupled service-based environments. Service compositions across ownership domains, promoted by SOA, call for addressing each one of the issues also from an inter-domain perspective.

Identification of dependency types The first challenge to be addressed is what kind of dependencies exist within a service-based application environment.
To realize the flexibility and adaptability of applications, SOA defines several service abstraction layers [6]. Thus, a service-oriented solution comprises different artifacts on a number of abstraction layers (cf. Figure 1). Processes coordinate business activities. A business activity is implemented by a service. Each service has implementation classes and description files to allow its discoverability. Relevant service contracts (SLAs) and test cases can also be of interest. Depending on the application scenario, only a subset of all existing dependencies have to be analyzed.

For the design of a dependency management solution, all types of artifacts comprising a service-based solution have to be identified. External services integrated into the system must also be considered as part of the configuration. Subsequently, the different kinds of dependency relations that exist between the identified artifacts have to be distinguished. To provide a generic dependency management framework, an extensible model allowing the representation of any type of dependency is needed.

**Selection of modeling approach** After all the possible dependency types have been identified, an appropriate way to represent relations of these types between artifacts is needed. A suitable modeling approach will have the capability of presenting all relevant information required for the documentation of the specified dependency variations. Since the configuration complexity of a service-based application landscape can get arbitrarily high, the model should allow the information presented to be restricted to the stakeholder’s needs.

Existing dependency modeling techniques should be examined against a list of requirements derived from the identified dependency types. A suitable model should be both understandable for humans and allow machine processing for automated analysis. Additionally, it should be powerful enough to express different dependency relations. Since an evolutionary change can be initiated both from business and IT, the representation should allow forward and backward traceability of the dependency information.
Dependency discovery and management The third challenge is the discovery of dependency information, since dependency relations are not explicitly defined [10]. Some dependencies can be found implicitly encoded. Others can be acquired from additional manual documentation. The challenging factor in the context of SOA is the large number of architectural artifacts which could be considered and the different formats that they are using for description. Additionally, in case of third-party services, users have limited access to service descriptions.

Regarding the acquiring and management of dependency information several decisions have to be made. The first one addresses the time for acquiring the data: during the design or the run-time of a system. Based on this first decision a different set of files will be available for extracting dependency information. The next decision is where to store the extracted data: a centralized or decentralized approach can be considered. Last but not least, an approach for keeping the dependency models up-to-date has to be developed. Both the collection and the management processes for dependency data have to be automated as much as possible to support the configuration representation of the complete system landscape and allow for independence of particular personal knowledge.

3 Research Approach

To address the research challenges specified in the previous chapter, the following work steps have been identified.

Identification of dependency types A two step approach will be applied to identify the types of dependencies which the generic model should be able to represent. First, existing reference architectures for SOA will be analyzed to identify all types of artifacts residing on the different SOA abstraction levels and considered a part of a SOA-based solution. Although SOA reference architectures do not give any guidance on dependency management, they provide, to different extents, information on the description of the separate architectural artifacts. Second, a number of modification scenarios will be defined and analyzed to extract the different types of dependencies that should be traced between the identified artifacts. Thereby we will consider bidirectional horizontal and vertical dependencies (within and across abstraction layers), and intra- and inter-domain dependencies, which consider the ownership domain boundaries between service providers and consumers. The results expected from this step include a dependency classification structure and a list of requirements for its representation.

Related work on dependency type identification considers only the second step of our approach. All existing solutions focus mostly on one application scenario. Most of the approaches consider either only horizontal [1, 2, 16, 10], or only vertical dependencies [4, 3, 5]. [15, 17] are handling both horizontal and vertical dependency relations. Only a few handle dependencies across domain boundaries [14, 3]. None of them considers a generic dependency type classification.
Selection of modeling approach  In the process of selecting a modeling approach, we will compare the properties of matrix-based, graph-based and formal models. A graph or matrix-based representation are easily understandable for humans, but rather limited in expression. Since all technologies currently used for the implementation of service-based systems use XML-based notations, we tend towards an XML-based approach for dependency information model. A formal XML-based notation is human readable and allows visualization as a graph. Automatic processing is granted by the variety of available XML-processing tools. Existing XML-query languages can be used to define selection criteria and provide a restricted view on the dependency data.

All approaches referenced here define their own formal notation, except for [16], [10] and [14] define also a XML-based schema for dependency information, which will be used as basis for the definition of our schema.

Dependency discovery and management  Considering life-cycle phases, dependency information can be collected during design or run-time. Some application domains like SLA management or new process composition rely on dependency information prior to execution. For this reason we will go for a design-time solution. Resources available in this phase include service- and process-descriptions, SLAs, and manual models. The information extracted from the different types of resources can be stored distributed. Dependency models, once computed for a specific application scenario, will be centrally saved for performance improvement. An update process will be triggered with every system modification to grant the freshness of the stored dependency relations.

Most of the existing approaches [15, 17, 4, 1, 5] select the design-time phase for the extraction of dependency information. Mainly used resources include service- and process-descriptions and SLAs [15, 17, 1]. [4] and [5] consider also manual models. Run-time solutions perform log analysis [2, 3]. The distribution of the collected data is addressed only in [10, 14]. All of them are lacking an approach for adaptable stakeholder specific dependency models representation.

4 Summary

How important is a service for providing business capabilities? How will its modification/withdrawal affect other applications? These questions have to be answered to ensure successful evolution of service-based infrastructures. The identification and documentation of dependencies for analysis purposes is a key factor for giving the answers. The contribution of this PhD proposal will be threefold: a generic dependency type classification; an understandable modeling notation allowing for automatic processing of dependency data; and a framework defining collection, documentation, and delivery process for dependency information. The described approach will provide for uniform representation of all types of dependency relations relevant for both design and maintenance needs. The explicit consideration of cross-domain relations will support the planning of adaptation needs even when modifications are triggered from external providers.
References
