Grant Agreement N° 215483

Title:	*QoS and SLA Aware Service Runtime Environment*
Authors:	*TUW, UOC, INRIA, SZTAKI*
Editors:	*Philipp Leitner , Harald Psaier (TUW)*
Reviewers:	*Pierluigi Plebani (POLIMI), Annapaola Marconi (FBK)*
Identifier:	*CD-JRA-2.3.9*
Type:	*Contractual Deliverable*
Version:	*1.0*
Date:	*28 Feb 2011*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable contains the final research outcomes of work package WP-JRA-2.3 (Self-* Service Infrastructure and Service Discovery Support). Hence, most focus is set on research in the area of service registries, autonomic service runtime environments and non-functional aspects of service-based systems. The deliverable is a paper-based document, integrating results from 7 individual research papers, authored by various members of WP-JRA-2.3. This deliverable outlines the individual research, and puts the conducted work in the broader context of the S-Cube framework, outlining clearly how the individual partner research relates to other work conducted in the work package, as well as to results produced in different parts of the S-Cube project. Additionally, we also given an outlook on open issues and future research directives, which have been opened up by the work presented here.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:

http://www.s-cube-network.eu/results/deliverables/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: http://www.s-cube-network.eu/

Contents

Chapter 1

Deliverable Overview

1.1 Introduction

This deliverable presents the final S-CUBE research outcomes in work package WP-JRA-2.3 (Self-* Service Infrastructure and Service Discovery Support). More concretely, this document presents final results of task T-JRA-2.3.2 (Service Registration and Search), which deals mainly with service infrastructures and service discovery. The goals of the deliverable, as stated in the most recent version of the S-CUBE Description of Work, are as follows:

> *CD-JRA-2.3.9: QoS and SLA aware service runtime environment [Month 48]: The main goal of this work is to propose a description of a novel service runtime infrastructure, which will incorporate an active and QoS-aware registry and client components. This infrastructure will ensure SLA compliance and suggest services as well as ad hoc processes.*

Hence, this deliverable will be a research- and paper-oriented document, focusing mainly on the topics of Quality-of-Service (QoS) management and service discovery based on non-functional properties in the context of service registries and runtime environments. To this end, this deliverable builds on CD-JRA-2.3.3 [38], which presented some groundwork requirements and research challenges. The current deliverable is meant as a continuation and implementation of this more vision-oriented earlier document. Additionally, the current deliverable has to be seen as complementary to PO-JRA-2.3.7 [66], which considered ad hoc process detection based on events. Finally, as the scope of this deliverable is very closely related to non-functional properties, QoS and Service Level Agreement (SLA) management, this deliverable in particular has strong links to S-CUBE work package WP-JRA-1.3 (End-to-End Quality Provision and SLA Conformance). More concretely, the notion of end-to-end SLAs, as incorporated here, is discussed in more detail in CD-JRA-1.3.3 [33], CD-JRA-1.3.4 [55] and CD-JRA-1.3.5 [63].

1.2 Deliverable Structure

The remainder of this document is structured as follows. In Section 1.3, we will revisit the research architecture of WP-JRA-2.3. This section gives a coarse-grained overview over the general research work carried out in the work package in total, and concisely summarizes what parts of the work package vision have been covered in this document. In Section 1.4, we will introduce the problem area of this deliverable. Most importantly, we will revisit the notions of Non-Functional Properties (NFP), Quality-of-Service (QoS), QoS-based service discovery, and Service Level Agreements (SLAs). The following Section 1.5 gives a brief introduction to the collected contributions, their relation to other deliverables, and other Workpackages, and future directions opened by the work . Afterwards, Chapter 2 will contain individual discussions and overviews over the contributed papers. These sections give a quick glance over the relevant research and integration between research results. Section 3.1 will conclude the main

part of the deliverable with a short summary, and an outlook on future research directions and remaining open issues. Finally, Appendix A contains the original papers in verbatim. This allows the interested reader to dig into the contributed research in all details.

1.3 The WP-JRA-2.3 Research Architecture

Research work in WP-JRA-2.3 is driven by the Work Package vision that structures the research work internally. Figure 1.1 illustrates the overall research architecture of WP-JRA-2.3: research on service infrastructures is comprised in three threads, Service Discovery, Service Registries and Service Execution. Orthogonally different approaches are separated in three layers.

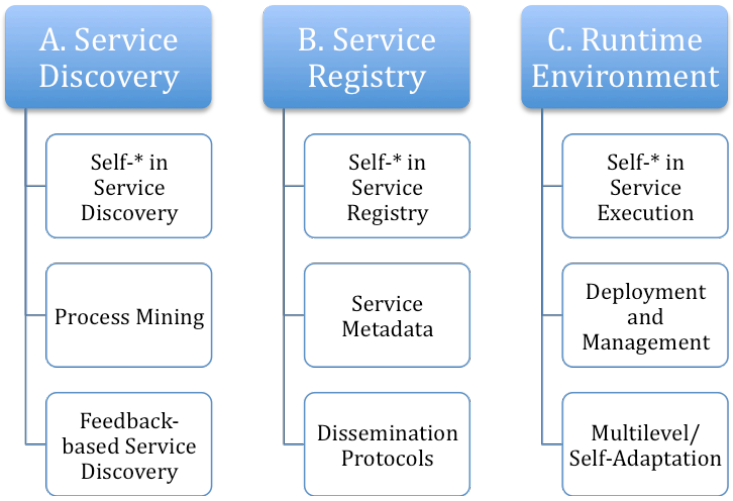


Figure 1.1: WP-JRA-2.3 Research Architecture

- Service Discovery Thread (A) - Service discovery is a fundamental element of service-oriented architectures, services heavily rely on it to enable the execution of service-based applications. Novel discovery mechanisms must be able to deal with millions of services. Additionally, these discovery mechanisms need to consider new constraints, which are not prevalent today, such as Quality of Experience requirements and expectations of users, geographical constraints, pricing and contractual issues, or invocability.
- Service Registry Research Thread (B) - Service registries are tools for the implementation of loosely-coupled service-based systems. The next generation of registries for Internet-scale service ecosystems are emerging, where fault tolerance and scalability of registries is of eminent importance. Autonomic registries need to be able to form loose federations, which are able to work in spite of heavy load or faults. Additionally, a richer set of metadata is needed in order to capture novel aspects such as self-adaptation, user feedback evaluation, or Internet-scale process discovery. Another research topic is the dissemination of metadata: the distributed and heterogeneous nature of these ecosystems asks for new dissemination methods between physically and logically disjoint registry entities, which work in spite of missing, untrusted, inconsistent and wrong metadata.
- Runtime Environment Research Thread (C) - There is an obvious need for automatic, autonomic approaches at run-time. As opposed to current approaches we envision an infrastructure that is able to adapt autonomously and dynamically to changing conditions. Such adaptation should be supported by past experience, should be able to take into consideration a complex set of conditions and their correlations, act proactively to avoid problems before they can occur and have a long lasting, stabilizing effect.

The current deliverable can be considered a cross-cutting, final discussion across all three research threads of the work package (service discovery, service registry and runtime environment). More concretely, the following aspects are covered by papers in this deliverable:

- We present some important contributions on research aspect A3 (feedback-based service discovery), related to skill-based discovery of human-provided services in marketplaces.
- In reserach thread B (service registry), we present important results on all three aspects (self-*, metadata and metadata dissemination). However, most focus is set on aspect B2 (service metadata).
- Finally, we also present results with regard to self-* in service execution (C1) and multi-level adaptation (C3), for instance within the scope of the PREvent framework.

However, please note that, as this is the final deliverable in this work package, the overal focus of the work presented was of an integrative nature, i.e., less focus was put on covering single aspects from the WP-JRA-2.3 research agenda, and more on cross-cutting research, spanning more than one aspect. Additionally, most research discussed in this deliverable also relates to other work packages of S-CUBE (most importantly, WP-JRA-1.2, WP-JRA-1.3 and WP-JRA-2.2).

1.4 Background

In the following section, we briefly summarize the most important background ideas used in the deliverable. This includes the idea of service metadata, non-functional service properties, Quality-of-Service, service discovery and Service Level Agreements (SLAs).

1.4.1 Non-Functional Properties and Quality-of-Service

Service descriptions include metadata about services. At least three different groups of metadata can be differentiated:

- *Functional descriptions* are the most common metadata, and define the functionality that a service provides. Simple functional descriptions can be published using WSDL [75]. More complex service descriptions are often specified using semantic Web service [47] technology, for instance WSMO [58] or SAWSDL [32]. However, there are also approaches which do not rely on semantics to provide more powerful functional descriptions [60].
- *Protocol descriptions* cover the dynamic aspects of service description. These are only relevant for stateful Web services, where service invocations have to be issued following a defined protocol or Message Exchange Patterns. These “usage protocols” for stateful services can be specified using languages such as BPEL Light [43] or the SEPL [20].
- Finally, QoS [48] aspects denote the non-functional properties of services. Hence, QoS is often seen as a discriminator between functionally equivalent services.

The abstract concept of QoS can be measured in virtually infinite different dimensions. Often-used dimensions to measure QoS are availability, response time, failure rate (reliability) and security. However, other research papers have identified a plethora of additional metrics. For instance, in [61] the dimensions wrapping time (time to unwrap the request XML structure), execution time (time necessary for the actual service invocation, excluding networking and message serialization issues) and network latency are proposed. Others consider trust and reputation as qualities that a service can exhibit [73, 68], and that can be used to differentiate between services.

1.4.2 Service Discovery Based on Non-Functional Properties

Service discovery is a process in which a service request is matched with the service descriptions/advertisements stored in a service registry. The result of this process is a set of service advertisements that match the request which may be grouped into a set of categories depending on their degree of match with this request. Based on the service aspect considered, this process can separated into two main sub-processes: a) functional service discovery, and b) non-functional service discovery. In the former process, the requester's functional requirements are matched with the functional capabilities of the registered services so as to infer those services that functionally match the request.

The non-functional service discovery process is usually called after the functional one. It is usually separated into two sequential sub-processes: a) non-functional service matchmaking, and b) service selection. The non-functional service matchmaking process filters the functional matching services based on their non-functional capabilities with respect to the non-functional requirements of the requester. It may also group the matching results based on their matching degree with the non-functional part of the service request. It must be noted here that both non-functional service capabilities and requirements are usually expressed as a set of constraints on non-functional properties and metrics. The service selection process then orders the matchmaking results according to their rank, which is usually produced through the Simple Additive Weighting [21] technique by considering requester-provided weights on non-functional terms (i.e., properties and metrics) as well as non-functional term-specific utility functions.

Research work in non-functional service matchmaking can be categorized into three main categories: ontology-based, constraint-based, and mixed. Ontology-based approaches [78] rely on semantic non-functional service specifications and use reasoners to infer the matching between non-functional service requests and advertisements. The main drawback of these approaches is that they are able to handle only n-ary constrained non-functional service specifications, i.e., specifications containing constraints on only one non-functional term. This drawback is solved by the constraint-based approaches [13, 11] which combine a non-functional service request and advertisement into one or more constraint models and use constraint solvers to solve these models and infer if there is a match between the non-functional request and advertisement. Depending on the linearity of the constraints involved in the specifications, different constraint solving techniques can be used [36]: a) Mixed-Integer Programming [65] for specifications containing only linear constraints and Constraint Programming [62] for specifications also containing non-linear constraints. However, such approaches assume that the non-functional service specifications reference specific non-functional models (i.e., descriptions of non-functional terms) that have been produced by one or more experts. The mixed-approach [36] relies on semantic non-functional service specifications and uses a specific algorithm [34] to align them based on their quality terms. It then follows the constraint-based approach to infer the matching of the specifications.

Apart from the previously described main approach in service selection, other sophisticated approaches [45] perform normalizations and grouping of non-functional metrics (in domains or functional groups). Normalizations are performed for three main reasons: a) to allow for a uniform measurement of service qualities independent of units, b) to provide a uniform index to represent service qualities for each provider, and c) to allow setting a threshold regarding the qualities. The number of performed normalizations depends on the nesting degree of the non-functional metric groups. All previous approaches rely on non-functional service advertisements that specify equality constraints on non-functional metrics (e.g. that the average response time is equal to 5 seconds). However, as in reality the non-functional service advertisements specify a range of values for each metric (i.e., two constraints defining the upper and lower value of a metric), approaches [11, 34] that solve Constraint Satisfaction Optimization Problems (SCOP) have been proposed to produce the rank for a non-functional service advertisement based on the worst (or even best) allowed value for the involved metrics.

1.4.3 Service Level Agreements

In a way, Web service SLAs [28] are a formalization and contractual arrangement of the concept of QoS. Instead of assuming that services provide the highest quality they can on a best-effort basis, SLAs fix the minimally promised quality in various dimensions. SLAs are often seen as legally binding contracts between one or more service clients and a service provider. SLAs are mainly a collection of SLOs. An SLO is an observable quality dimension of a service. Evidently, there is a strong relationship between SLOs and QoS, and, indeed, frequently SLOs are defined on top of QoS characteristics. For instance, often-used SLOs are the response time and availability of the service. Additionally, SLAs define penalties for non-achievement (violation) of SLOs. Penalties are often monetary consequences, which are expected to press service providers to achieve their target values. Both, penalties and target values, can be different for every SLA in which a an SLO is used. At runtime, concrete values for SLOs can be monitored. Based on the type of SLO (see below), this measured value can be generated either per composition instance or per aggregation interval.

Some different languages have been proposed to model SLA, including WSLA [12, 28], WS-Agreement [3] and SLAng [67]. These models do not differ so much in their expressiveness, but more in the environment they live in. For instance, WSLA specifies a monitoring and accounting infrastructure along with the basic language [12]. It is important to note that the work in this deliverable is agnostic with regard to the used SLA language, as long as it fits the basic model described above.

Types of SLOs

Just like QoS dimensions, SLOs come in different flavors. In this deliverable, two distinctions are of relevance.

- Firstly, one can differentiate between nominal and continuous SLOs. For nominal SLOs, the measured value of an objective can be one of a finite number of potential values. For these SLOs, the target value is a subset of the set of potential values. Metric SLOs, which are more prevalent, can take an infinite number of values. Target values are defined as thresholds on the metric.

- Secondly, one can distinguish SLOs on composition instance level and aggregated SLOs. For composition instance level SLOs, a decision of whether an SLA violation has happened can be made for every single composition instance individually. Aggregated SLOs are defined over an aggregation period, for instance a number of composition instances or a time interval. Decisions can be made only looking at the whole aggregation period, i.e., usually numerous composition instances. Unless stated otherwise, all work in this deliverable is applicable for composition instance level SLOs only. A generalization of the presented approach to aggregated SLOs is part of ongoing work.

1.5 Overview of the Contributions

In the following, we briefly introduce the contributions of the deliverable. More detailled summaries can be found in Chapter 2.

1.5.1 Stimulating Skill Evolution in Market-based Crowdsourcing [64]

Content Overview. This work presented at the 9th International Conference on Business Process Management (2011) presents Crowdsourcing on top of a SOA related infrastructure A major challenge in crowdsourcing is to guarantee high-quality processing of tasks. The work presents a novel approach that matches tasks to suitable workers based on auctions. This way QoS constraints can be better met and agreements settled with customers.

Relations to 2.3. The work is related to platform, thus, infrastructure management (JRA-2.3). In particular, the content can be seen as an extension to the ranking approaches in CD-JRA-2.3.5. Instead of discussing ideas of local approaches such as in CD-JRA-2.3.4 and CD-JRA-2.3.6 here a more global approach is taken when adapting the environment's resource selection. The adaptation strategies presented do not consider direct interference with the infrastructure but instead choose an offline approach that feeds its decision information from interaction observations, e.g., skill evolution.

Relations to other workpackages. While the current version of the paper has little relations to other work packages, future workcould go in the direction of compositions as studied by JRA-2.2. In particular, coordinated service compositions as researched in CD-JRA-2.2.2 and algorithms and techniques for splitting and merging compositions as presented in CD-JRA-2.2.3 would then be in the focus of the studies.

Future Directions. As part of the ongoing research the plan is to investigate the difficulties to introduce complex tasks and in the crowd collaboration. Sub-tasks need to be decompose and reassembled. The responsibility for a certain QoS would also transfer partially to the crowd members.

1.5.2 End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo [50]

Content Overview. Published in IEEE Transactions on Services Computing, this work copes with the challenge of dynamic and adaptable service-oriented solutions with special emphasis on service metadata, Quality of Service, service querying, dynamic binding and service mediation. The Vienna Runtime Environment for Service-oriented Computing (VRESCo) is presented in the light of a service infrastructure with details on service querying and service mediation.

Relations to 2.3. This work is integral to to research agenda of JRA-2.3 (service registries), as it presents a novel service registry model, which is particularly suitable to support adaptive SOAs. Service querying in VRESCo builds on the ranking approaches discussed in CD-JRA-2.3.5.

Relations to other workpackages. As VRESCo is, by design, QoS-aware, this contribution also strongly relates to JRA-1.3, most importantly to the QoS evaluation methods discussed in CD-JRA-1.3.4, CD-JRA-1.3.5 and CD-JRA-1.3.6.

Future Directions. Future work within VRESCo will see us focus more on support and integration of service compositions and business processes. Hence, this line of research will mostly continue within the topics covered by JRA-2.2.

1.5.3 Cost-Based Optimization of Service Compositions [40]

Content Overview. This work, accepted for publication in IEEE Transactions on Services Computing, tackles the challenge of preventing cases of SLA violations for providers of composite services. In order to get a realistic and complete view of the decision process of service providers, the costs of adaptation need to be taken into account. The solution presented offers possible algorithms to solve this complex optimization problem, and details an end-to-end system based on the PREvent (prediction and prevention based on event monitoring) framework, which clearly indicates the usefulness of the model.

Relations to 2.3. At its core, this contribution is discussing an autonomic service-based system for managing customer SLAs. To this end, we use the VRESCo QoS-aware service registry (see previous contribution) as foundation. Hence, this contribution relates to the self-* service registry and infrastucture part of JRA-2.3.

Relations to other workpackages. Evidently, this paper also has strong relationships to JRA-1.2, especially CD-JRA-1.2.4, CD-JRA-1.2.5 and CD-JRA-1.2.6. Furthermore, the contribution relates strongly to the SLA topics discussed in JRA-1.3 (CD-JRA-1.3.6). Finally, as the main subject of adaptation in this paper are service compositions, the paper also relates to JRA-2.2 (CD-JRA-2.2.6).

Future Directions. There is still potential for plenty of further research in the direction of cost-based optimization. For instance, in its current form, the formalization used in the paper does not take into account indirect costs, such as customer satisfaction or potential loss of future customers. Furthermore, the current version of the paper suffers from the limitation that it considers only SLAs on instance-level. Future work can extend and improve on those aspects.

1.5.4 Towards Optimizing the Non-Functional Service Matchmaking Time

Content Overview. Published as a poster contribution at the The World Wide Web Conference, the approach concentrates on exploiting constraint solving techniques in service discovery for inferring if the user non-functional requirements are satisfied by the service non-functional capabilities. This paper proposes two alternative techniques for improving the non-functional service matchmaking time. The first one is generic as it can handle non-functional service specifications containing n-ary constraints, while the second is only applicable to unary-constrained specifications.

Relations to 2.3. The research work addresses the JRA-2.3's research challenge that concerns scalable and fault tolerant techniques for service discovery as it proposes two QoS-based service matchmaking techniques which not only optimize the matchmaking time without sacrificing accuracy but they can also be distributed through assigning part of the matchmaking functionality to different nodes.

Relations to other workpackages. The research work is highly connected to the JRA-1.3 WP as it exploits non-functional service descriptions (requests as well as advertisements), which could be described through the non-functional service meta-model proposed in the deliverable CD-JRA-1.3.3 and could reference the non-functional properties of the S-Cube's end-to-end quality reference model proposed in the deliverable CD-JRA-1.3.2, so as to perform the matchmaking.

Future Directions. The research work can be exploited by scalable service registries which are able to matchmake millions of services, paving the way for the move towards the Internet of Services.

1.5.5 Cost Reduction Through SLA-driven Self-Management [37]

Content Overview. Presented at the 9th IEEE European Conference on Web Services, the presented approach considers a SLAs management which satisfies the customers requirements and also their own business objectives, such as maximizing profits. Most current systems fail to consider business objectives and thus to provide a complete SLA management solution. Specifically, this work proposes a framework that comprises multiple, configurable control loops and supports automatically adjusting service configurations and resource usage in order to maintain SLAs in the most cost-effective way. The framework targets services implemented on top of large-scale distributed infrastructures, such as clouds.

Relations to 2.3. The work is placed in the context of *Self-* in Service Execution*. Qu4DS provides an automatic support for executing services by using self-adaptive techniques (dynamic adaptation). With regard to *Deployment and Management*, the Qu4DS framework automatically deploys service instances on top of distributed infrastructures by managing them according to quality properties.

Relations to other workpackages. The work is complementary to adaptable service compositions (i.e., WP-JRA-2.2) as it prevents SLA violations thus avoiding triggering adaptation actions in the

composition-level. Moreover, composition-level adaptable techniques can be aware about further details about Qu4DS adaptive behavior which enables to conceive multilevel adaptation. With respect to the deliverable CD-JRA-2.3.2, the work can be considered as a self-healing support for atomic services. With respect to the deliverable CD-JRA-2.3.6, Qu4DS specifies and employs adaptation policies but not limit to them, the framework can be extended to support further adaptation policies.

Future Directions. The work leaves place for several research directions. First, the QoS translation can be improved based on advanced application profiling techniques. This may require estimating application performance based on profiled data at runtime for instance. Second, Qu4DS could rely on dynamic pricing where the price of the service take into account further aspects, for example, related to competitor service prices. Third, other service providers can be supported, not only those which rely on distributed tasks. This requires further dynamic metrics which are able to analyze if the request treatment is in time or delayed for example. Finally, further adaptation policies can be developed not only in the sense of the current addressed events (request arrivals and job faults and delays), but also in the context of other events such as contract proposals, resource failures, infrastructure price changes, contract rescission and so forth.

1.5.6 Autonomic SLA-aware Service Virtualization for Distributed Systems [30]

Content Overview. Presented at the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, the focus in this work is on Cloud Computing. Managing such heterogeneous environments requires sophisticated interoperation of adaptive coordinating components. The work introduces an SLA-aware Service Virtualization architecture that provides non-functional guarantees in the form of SLAs and consists of a three-layered infrastructure including agreement negotiation, service brokering and on demand deployment. In order to avoid costly SLA violations, flexible and adaptive SLA attainment strategies are used with a failure propagation approach.

Relations to 2.3. The work introduced in this paper is a result of a SZTAKI-TUW collaboration in the Runtime Environment research thread of JRA-2.3. It targets the management and deployment aspects of interoperating distributed systems, and proposes autonomic SLA management strategies to deal with the highly dynamic and error-prone nature of these systems. In the S-Cube deliverable CD-JRA-2.3.2, a conceptual architecture incorporating three closely related areas was introduced: agreement negotiation, service brokering and service deployment. The examinations of this architecture revealed the basic requirements for a future self-* realization of these core components. These basic requirements implied that there must be a negotiation phase when it is specified, what service is to be invoked and what are the conditions and constraints (temporal availability, reliability, performance, cost, etc.) of its use. The contribution to Deliverable CD-JRA-2.3.4 refines this vision, and presents a unified service virtualization environment representing the first attempt to combine SLA-based resource negotiations with virtualized resources in terms of on-demand service provision resulting in a holistic virtualization approach. The contributino to Deliverable CD-JRA-2.3.6 introduces the autonomic behaviour of this service virtualization architecture by summarizing the same contribution. The focus is on the SLA-awareness of the architecture.

Relations to other workpackages. The topic of this contribution is also closely related to research carried out in WP-JRA-1.2 and WP-JRA-1.3. Detailed adaptation-related capabilities of the architecture including cross-layer adaptation were reported in deliverable CD-JRA-1.2.5.

Future Directions. In this current contribution a refined architecture was introduced considering resource provision using a virtualization approach combined with the business-oriented utilization. This solution utilizes a heterogeneous SLA-coupled infrastructure for on-demand service provision based on SLAs with a MAPE-based autonomic behaviour, in order to cope with changing user requirements and on demand failure handling. Since newly emerged technologies such as Cloud Computing become

more and more utilized in business service solutions, the extension of the proposed SSV solution for additional Cloud infrastructures and platforms will require further research.

The volume of works collected in this deliverable present different approaches to the subject of the deliverable. The particular challenges studied in the partners' contributions describe approaches taken from different perspectives and taking into account various relevant aspects of the Service-Oriented Architecture (SOA). In the next section, all partners present their contribution in the scope of this deliverables description.

Chapter 2

Contributions to QoS and SLA aware service runtime environment

2.1 Stimulating Skill Evolution in Market-based Crowdsourcing

Contributing partners: Vienna University of Technology (TUW)

Status: Submitted to the 9th International Conference on Business Process Management (BPM) , 28th August - 2nd September, 2011, Clermont-Ferrand, France.

Keywords: Human-centric BPM, Crowdsourcing, Online communities, Task markets, Auctions, Skill evolution

2.1.1 Background

Today, ever changing requirements force in-house business processes to rapidly adapt to changing situations in order to stay competitive. Changes involve not only the need for process adaptation, but also, require an additional inclusion of new capabilities and knowledge, previously unavailable to the company. Thus, outsourcing of parts of business processes became an attractive model. This work, in particular, focuses on a distinguished recent type of outsourcing called *crowdsourcing*. The term crowdsourcing describes a new web-based business model that harnesses the creative solutions of a distributed network of individuals [8], [72]. This network of humans is typically an open Internet-based platform that follows the *open world* assumption and tries to attract members with different knowledge and interests. Large IT companies such as Amazon, Google, or Yahoo! have recognized the opportunities behind such *mass collaboration systems* [14] for both improving their own services and as business case. In particular, Amazon focuses on a task-based marketplace that requires explicit collaboration. The most prominent platform they currently offer is *Amazon Mechanical Turk* (AMT) [2]. Requesters are invited to issue *human-intelligence tasks* (HITs) requiring a certain qualification to the AMT. The registered customers post mostly tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [22]).

2.1.2 Problem Statement

In this work we assume that crowdsourcing can be build on top of a SOA environment. The main challenges addressed in this work relate to building and managing an automated crowd platform. There is only a few approaches towards integrating SLAs into crowdsourcing. In one of our previous works at TUW [57] we introduced an approach to include human tasks as available on the current crowd platforms into Web Service Level Agreements (WSLA). The interested reader is referred to the results of that

work for a more elaborated study of the idea. This present work relates more to QoS related issues. In crowdsourcing it is not only of importance to find suitable workers for a task and to provide the customer with satisfying quality, but also, to maintain a motivated base of crowd members and provide stimulus for learning required skills. Only a recurring, satisfied crowd staff is able to ensure high QoS and high output. As any crowd, fluctuations must be compensated and a *skill evolution* model must support new and existing crowd workers in developing their capabilities and knowledge. Finally, the standard processes on such a platform should be automated and free from intervention to handle the vast amount of tasks and to make it compatible with a SOA approach. Atop, the model should increase the benefit of all participants to support QoS.

2.1.3 Contribution Relevance

As outlined previously, a further major challenge hampering the establishment of a new service-oriented computing paradigm spanning enterprise and open crowdsourcing environments are quality issues. In our scenario this is strongly connected to correctly estimating the skills of workers. Thus, the presented **skill evolution** approach helps to increase the confidence in worker skills with qualification tasks. This would imply a huge overhead for the testing requester; s/he is also the only one who benefits from the gathered insights. Here, we take a different approach by integrating the capability of confidence management into the crowdsourcing platform itself. Instead of having point-to-point tests, we propose the automated assessment of workers to unburden requesters in inspecting workers' skills. The approach is *(semi-)automatic* and offers great potential for inclusion of crowd capabilities in business environments. Knowing crowd capabilities by *skill evolution* is one of the major steps towards better QoS in an open crowdsourcing environment.

2.1.4 Contribution Summary

Our idea of skill evolution contains the following steps. We propose the *automatic assessment* of workers where confidence values are low. For example, newcomers who recently signed up to the platform may be high or low performers. To unveil the tendency of a worker, we create a hidden 'tandem' task assignment comprising a worker whose skills are known (high performer) with a high confidence and a worker where the crowdsourcing platform has limited knowledge about its skills (i.e., low confidence). The next step is that both workers process the *same* task in the context of a requester's (real) task. However, only the result of the high confidence worker is returned to the requester, whereas the result of the low confidence worker is compared against the delivered reference.

This approach has advantages and drawbacks. First, skill evolution through tandem assignments provides an elegant solution to avoid training tasks (assessments are created automatically and managed by the platform) and also implicitly stimulates a learning effect. Of course, the crowdsourcing platform cannot charge the requester for tandem task assignments since it mainly helps the platform to better understand the true skill (confidence) of a worker. Thus, the platform must pay for worker assessments. As the evaluations show, performing assessments provides the positive effect that *the overall quality* of provided results and thus requester *satisfaction increases* due to a better understanding of worker skills.

2.1.5 Contribution Evaluation

To evaluate the ideas of the work we have implemented a Java-based simulation framework that supports all introduced concepts. An evaluation scenario consists of a set of workers and a set of requesters. In every round of the simulation each requester usually announces a task. An auction is conducted for each announced task which contains among other properties the expected quality. High quality requirements indicate highly sophisticated and demanding tasks. We applied different scenarios and tested those with and without skill evolution. Detailed description of the setup is available in the work's section on implementation an evaluation.

The results show that the additional assessments provide remarkable good results for reducing misjudgements. As a summary, skill evolution generally performs better, however, is not antagonistic to overload scenarios. While with moderate task offering frequencies the model performs much better in all measurements, the differences become even when load increases and assessment task further overload the platform. The results show that it is the responsibility of the platform to balance the task load and trade only with a fair amount of requesters.

2.1.6 Conclusions

In the contribution we present a novel crowdsourcing marketplace based on auctions. The design emphasizes automation and low overhead for users and members of the crowdsourcing system. In oder to increase the QoS in SOA-based crowdsourcing we introduce an approach with two novel auctioning variants and show by experiments that it may be beneficial to employ assessment task in order to estimate members' capabilities and to train skills. Stimulating the demand for certain skills in such a way leads to skill evolution. From such an evolution can benefit all participants. New workers can be motivated by helping them to develop their skills with training. This way the platform provider can harvest a regular group of returning workers. The customers are also satisfied because the quality of the service because the auctioning mechanism guarantees that the best fitting worker gets their tasks.

2.2 End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo

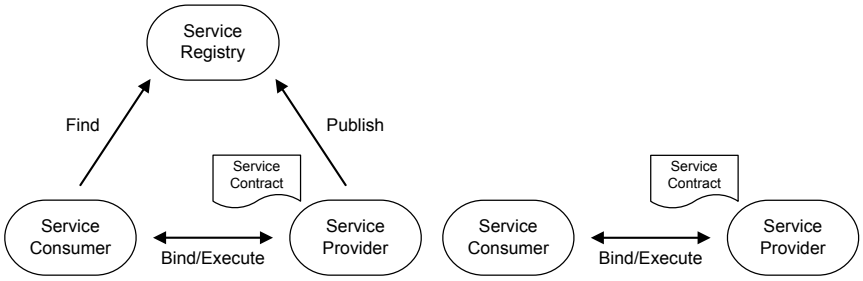
Contributing partners: Vienna University of Technology (TUW)

Status: Published in IEEE Transactions on Services Computing (2010)

Keywords: Web Services Publishing and Discovery, Metadata of Services Interfaces, Advanced Services Invocation Framework

2.2.1 Background

During the last years, Service-oriented Architecture (SOA) and Service-oriented Computing (SOC) have gained acceptance as a paradigm for addressing the complexity that distributed computing generally involves. In theory, the basic SOA model consists of three actors that communicate in a loosely coupled way as shown in Figure 2.1a. However, practice has shown that SOA solutions are often not as flexible and adaptable as claimed. We argue that there are some issues in current implementations of the SOA model. First and foremost, service registries such as UDDI did not succeed. We think this is partly due to their limited querying support that only provides keyword-based matching of registry content, and insufficient support for metadata and non-functional properties of services. This is also highlighted by the fact that Microsoft, SAP, and IBM have finally shut down their public UDDI registries in 2005. As a result, service registries are often missing in service-centric systems (i.e., no *publish* and *find* primitives). This leads to point-to-point solutions where service endpoints are exchanged at design-time (e.g., using E-mail) and service consumers statically bind to them (see Figure 2.1b). Besides that, support for dynamic binding and invocation of services is often restricted to services having the same technical interface. In this regard, the lack of service metadata makes it difficult for service consumers to know if two services actually perform the same task. Furthermore, support for Quality of Service (QoS) is necessary to enable service selection based on non-functional QoS attributes such as response time (in addition to functional attributes).



(a) SOA Model (b) SOA Practice

Figure 2.1: SOA Theory vs. Practice (adapted from [51])

2.2.2 Problem Statement

Adaptive service-oriented systems bring along several distinct requirements, leading to a number of challenges that have to be addressed. In this section, we summarize the current challenges we see most important, and which are addressed in the paper summarized here. (1) Firstly, service interface description languages such as WSDL focus on the interface needed to invoke a service. However, from this interface it is often not clear what a service actually does, and if it performs the same task as another service. Service metadata [7] can give additional information about the purpose of a service and its interface (e.g., pre- and post-conditions). (2) Once services and associated metadata are defined, this information should be discovered and queried by service consumers. This is the focus of service registry standards such as UDDI. In practice, the service registry is often missing. (3) In enterprise scenarios, QoS plays a crucial role [77] in discriminating between services. This includes both network-level attributes (e.g., latency and availability), and application-level attributes (e.g., response time and throughput). (4) Finally, in order to technically enable actual dynamicity and runtime service binding, mechanisms that mediate between alternative services, possibly having different interfaces, need to be provided.

2.2.3 Contribution Relevance

In order to tackle the four challenges outlined above, we have devised and implemented a service runtime environment called VRESCo, which aimed at providing a practical, integrative solution to adaptive enterprise services computing. To be more specific, the present paper focuses on service metadata, QoS and service querying, plus dynamic binding, invocation, and mediation of services. Additionally, we provide an extensive performance evaluation of the different components and an end-to-end evaluation of the overall runtime, that shows the applicability of our approach.

2.2.4 Contribution Summary

The architectural overview of VRESCo is shown in Figure 2.2, which is adapted from [49]. The VRESCo core services are provided as Web services that can be accessed either directly using SOAP or by using the Client Library that provides a simple API. Furthermore, the DAIOS framework [41] has been integrated into the Client Library, and provides stubless, protocol-independent, and message-driven invocation of services. The Access Control Layer guarantees that only authorized clients can access the core services, which is handled using claim-based access control and certificates [49]. Services and associated metadata are stored in the Registry Database which is accessed using the Object-Relational Mapping (ORM) Layer. Finally, the QoS Monitor is responsible for regularly measuring the current QoS values. The overall system is implemented in C# using the Windows Communication Foundation [46]. Due to the platform-independent architecture, the Client Library can be provided for different platforms (e.g., C# and Java).

There are several core services. The Publishing/Metadata Service is used to publish services and metadata into the Registry Database. Furthermore, the Management Service is responsible for managing

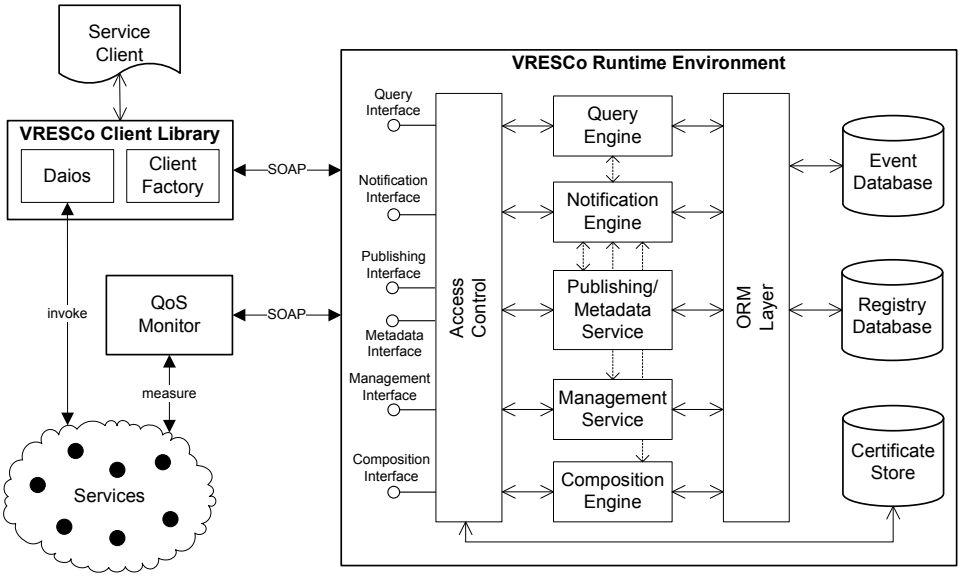


Figure 2.2: VRESCo Overview Architecture

user information (e.g., name, password, etc.) whereas the Query Engine is used to query the Registry Database. The Notification Engine informs users when certain events of interest occur inside the runtime, while the Composition Engine [59] provides mechanisms to compose services by specifying hard and soft constraints on QoS attributes. In this paper, we focus on the main requirements for our client-side mediation approach which are the Metadata Service (including the models for metadata, services and QoS), the Query Engine, and the dynamic binding, invocation and mediation mechanisms.

2.2.5 Contribution Evaluation

We give an evaluation of the VRESCo runtime focusing on the topics covered in this paper. The purpose of this evaluation is twofold: firstly, we show the runtime performance regarding service querying, rebinding, and mediation by using synthetic data. The main goal of this evaluation is to analyze the performance impact of each aspect in isolation. Secondly, we combine these aspects into a coherent end-to-end evaluation using an order processing workflow. The main goal is to understand the influence of each aspect with regard to the overall process duration in a realistic setting. Additionally, we show how the individual results of the first part interrelate in an end-to-end setting. For mediation, rebinding and end-to-end evaluation we have created different sets of test services and QoS configurations (with varying response times) using the Web service generation tool GENESIS [26].

2.2.6 Conclusions

One of the main promises of SOC is the provisioning of loosely-coupled applications based on the publish-find-bind-execute cycle. In practice, however, these promises can often not be kept due to the lack of expressive service metadata and type-safe querying facilities, explicit support for QoS, as well as support for dynamic binding and mediation. In this paper, we have proposed the QoS-aware VRESCo runtime environment which has been designed with these requirements in mind.

2.3 Cost-Based Optimization of Service Compositions

Contributing partners: Vienna University of Technology (TUW)

Status: Accepted for publication in IEEE Transactions on Services Computing (2012)

Keywords: Service Composition, Service Level Agreements, Adaptation, Optimization

2.3.1 Background

Service-based applications have seen tremendous research activity in the last years, with many important results being generated around the world. However, to fully realize its potential, research and industry alike need to focus more strongly on non-functional properties and quality issue of services. In the business world, QoS promises are typically defined within legally binding Service Level Agreements (SLAs) between clients and service providers, represented, e.g., using WSLA. SLAs contain Service Level Objectives (SLOs), i.e., concrete numerical QoS objectives, which the service needs to fulfill. If SLOs are violated, agreed upon monetary consequences go into effect. For this reason, providers generally have a strong interest in monitoring SLAs and preventing violations, either by using post mortem analysis and optimization [74], or by runtime prediction of performance problems [39]. We argue that the latter is more powerful, allowing to prevent violations before they have happened by timely application of runtime adaptation actions.

2.3.2 Problem Statement

However, preventing SLA violations is, in general, not for free. For instance, some alternative services usable in a composition may provide faster response times (thereby improving the end-to-end runtime of the composite service, and reducing the probability of violating runtime related SLOs), but those services are often more expensive than slower ones. Therefore, there is an apparent tradeoff between preventing SLA violations and the inherent costs of doing so. We argue that this tradeoff is currently not covered sufficiently in research. Instead, researchers assume that the ultimate goal of service providers is to minimize SLA violations, completely ignoring the often significant costs of doing so (e.g., [42, 27]).

2.3.3 Contribution Relevance

In this paper, we contribute to the state of the art by formalizing this tradeoff as an optimization problem, with the goal of minimizing the total costs (of violations and applied adaptations) for the service provider. We argue that this formulation better captures the real goals of service providers. Additionally, we present possible algorithms to solve this optimization problem efficiently enough to be applied at composition runtime.

2.3.4 Contribution Summary

In the paper, we model the decision process of a service provider as in Equation 2.1. Therein, TC are the total costs for the service provider; S contains all SLOs contained in the SLA of the provider; e^i_{sx} is an estimation function capturing the predicted costs for SLA violations; A^* is the set of applied adaptation actions (from the set A of potential adaptations); $c(a_x)$ are the costs of applying adaptation action a_x; finally, $v(A^*)$ is a penalty term that captures whether A^* contains any conflicting adaptation actions. For more details, please refer to the original paper.

$$TC(A^*) \approx v(A^*) + \sum_{s_x \in S} e^i_{sx} + \sum_{a_x \in A^*} c(a_x) \rightarrow min! \tag{2.1}$$

In addition to the optimization problem formalization, we also discuss different approaches for finding solutions to this problem. Firstly, we present a deterministic branch-and-bound approach, which is guaranteed to find the optimal solutions. However, given the problem structure and the generally very large solution space, for many problem instances a deterministic solution is unfeasible. For these cases,

we present two alternative heuristic approaches: a local search algorithm, which finds solutions very efficiently, and a genetic algorithm based approach, which takes more time to find solutions but is able to evade local optima.

2.3.5 Contribution Evaluation

In the paper, we evaluated our approach based on an illustrative example, which has been implemented using .NET Windows Communication Foundation[1] (WCF) technology and the VRESCo SOA runtime environment on a server running Windows Server Enterprise 2007, Service Pack 2. The machine was equipped with 2 2.99GHz Xeon X5450 processors and 32 GByte RAM. More details on the experimental setup can be found in the accompanying experimentation web page[2].

Drawing conclusions from our experiments, we note that Branch-and-Bound is applicable in situations where just a small set of actions is available. If more actions are available, Memetic Algorithms and GRASP are interesting candidate algorithms. GRASP produces good solutions in very little time and can generally be used even for short-running compositions where adaptation decisions need to be taken in a short time frame (below 1 second). Memetic Algorithms are very promising in case of long-running compositions, where the time necessary to find a solution is not critical. Memetic Algorithms often produce slightly better solutions than GRASP, but take much more time to do so. In a second set of experiments, we also evaluated the end-to-end effectiveness of our system. That is, we analyze if the system fullfills its main promise, preventing SLA violations and reducing the total costs for the service provider. As can be seen in the paper, our system fulfills its main promise: in the example case, the total number of SLO violations decreases to about 28% of the number of predicted violations. However, we can also see that it does not primarily prevent violations, but rather aims at minimizing the costs of violations. Thereby, the total costs for the service provider can be reduced to 56% of the predicted costs.

2.3.6 Conclusions

For providers of composite Web services, it is essential to be able to minimize cases of SLA violations. One possible route to achieve this is to predict at runtime, which instances are in danger of violating SLAs, and to apply various adaptation actions to these instances only. However, it is not trivial to identify which adaptations are the most cost-effective way to prevent any violation, or if it is at all possible to prevent a violation in a cost-effective way. In this paper, we have modelled this problem as a one-dimensional, discrete optimization problem. Furthermore, we have presented both, deterministic and heuristic solution algorithms. We have evaluated these algorithms based on a manufacturing case study, and shown which types of algorithms are better suited for which scenarios.

2.4 Towards Optimizing the Non-Functional Service Matchmaking Time

Contributing partners: University of Crete

Status: conditionally accepted (as a poster paper) at WWW'12

2.4.1 Background

One of the main drivers and mechanisms towards the Internet of Services (IoS), where millions of services will be available to users through a converged information, communication, and service infrastructure, is service-orientation as it promises the automatic construction of novel, added-value applications through the discovery and composition of services. However, service-orientation has not yet kept up

[1] http://msdn.microsoft.com/en-us/library/ms735967(VS.90).aspx
[2] http://www.infosys.tuwien.ac.at/prototype/VRESCo/experimentation.html

to its promises as it relies on a specific architecture, in which the service broker role, associated to the discovery of services, has not been successfully fulfilled by the current implementations. Such implementations lack the appropriate scalability that is required for matching millions of services and rely on service discovery mechanisms that are not very efficient in terms of matchmaking time and accuracy.

Concerning the functional service discovery, the state-of-the-art research work [10, 56, 31] relies on using semantic I/O-based service descriptions and exploits Semantic Web techniques to perform the functional service matchmaking. It has been shown that such work has a very good matchmaking time but not a perfect accuracy as it does not rely on the service goals, expressed by service pre- and post-conditions, as such descriptions are not yet provided by the service providers. Another line of work [16, 69, 52, 35] has focused on scalability issues and on further optimizing the matchmaking time by appropriately organizing the service advertisement and query space.

2.4.2 Problem Statement

While the research status concerning functional service discovery is satisfactory, the same cannot be stated for non-functional service discovery. The state-of-the-art approaches [11, 36] in the latter sub-area exhibit perfect accuracy and have mainly focused on optimizing the time required to matchmake a single non-functional request-to-advertisement pair by exploiting appropriate constraint solving techniques. However, they have not yet focused on optimizing the overall non-functional matchmaking time. To this end, they can take significant time to match a set of hundreds or thousands of non-functional service advertisements against a non-functional service request, so they are not yet appropriate for the move to the IoS era. In addition to the above problem, there have not been approaches focusing on scalability issues.

Thus, there is a need for scalable non-functional service discovery techniques that can appropriately manage a vast amount of non-functional service advertisements and optimize the time needed to match them against non-functional service requests. If such techniques were coupled with those proposed in functional service discovery, then a better service broker implementation would have been realized, which could enable users and automated agents to discover those services that perfectly match their tasks both functionally and non-functionally in a timely manner and with the appropriate, if not perfect, accuracy.

2.4.3 Contribution Relevance

By exploiting the matchmaking metric of the approach in [36], this work partially closes the above gap by proposing two different matchmaking techniques which are able to optimize the overall non-functional service matchmaking time without sacrificing matchmaking accuracy. This is shown both theoretically and empirically by comparing these two novel techniques against the most prominent state-of-the-art one proposed in [36] in terms of matchmaking time. In fact, one of these two techniques, called ”Unary matching” technique, is shown to be far better than the other and the prominent one not only concerning matchmaking but also insertion, deletion, and update time. Another significant advantage of the proposed work is that both techniques can be easily distributed in order to realize scalable matchmaking mechanisms.

2.4.4 Contribution Summary

The first proposed technique, called “Subsumes matching” technique, relies on the “subsumes” type of matchmaking metrics and on the fact that if a non-functional service specification A subsumes another specification B then it will also subsume all the specifications that are subsumed by B. To this end, it organizes the non-functional service advertisement space in such a way that the number of non-functional request-to-advertisements pairs examined is less than that of the state-of-the-art approach. In particular, it constructs a forest of “subsumes” trees, where each node corresponds to a non-functional service

advertisement and a parent node in each tree subsumes all of its children nodes. In this way, when a service request is issued, it is compared against the nodes of each tree from the root until the leaves. However, if it is found that it subsumes a specific node, then there is no need to go further down to the node's children/descendants as the request will certainly match/subsume them.

It is obvious that this technique is quicker than the state-of-the-art one, as each one uses the same matchmaking metric but the first technique performs less comparisons. However, the construction and update of a forest of "subsumes" trees is more costly than the construction and the update of a list of service advertisements (as it is the case for the prominent approach). To this end, this technique exhibits a higher insertion, deletion, and update time with respect to the prominent approach. This technique can be distributed by assigning the responsibility of matching a subset of the subsumes trees to different nodes.

The second proposed technique (i.e., the "Unary matching" one) relies on similar techniques performed in functional service matchmaking [10] in order to appropriately organize the non-functional service advertisement space. In particular, it maintains for each non-functional metric/property an ordered set of limits, where each limit may correspond to one or more non-functional specifications containing a respective metric bound or equality constraint on the limit's value. To this end, when a non-functional service request is issued, each of its unary constraints are examined based on their containing metric. Depending on the metric bound and constraint type, a sub-part of the metric's ordered list of limits is examined so as to produce a list of the matching non-functional advertisements' URIs. For instance, if the request constraint is of the form: $X \leq a$, then the limits that are equal or less to a are examined and the URIs of the non-functional specifications that have constraints of the form $X \leq a_1$, or $X == a_1$, where $a_1 \leq a$, are collected. For each request constraint, its constructed URI list is concatenated with that of the previous constraint. If the URI list concatenation is empty, then the non-functional request does not have a matching advertisement. Otherwise, after the processing of the last request constraint, the final, concatenated URI list contains all the URIs of the advertisements that match the request.

The second technique is quicker than the other proposed technique as well as the prominent matchmaking approach not only in terms of matchmaking but also insertion, deletion, and update time. Moreover, due to the way it organizes the advertisement space, it can be easily distributed by assigning the responsibility of matching a sub-set of all non-functional metrics to different nodes. Its sole drawback is that it relies on exploiting only unary-constrained non-functional service specifications.

2.4.5 Contribution Evaluation

The two proposed techniques were both theoretically and empirically evaluated against the most prominent non-functional service matchmaking approach [36]. The results reported validate the comparison statements referenced in the previous subsection. In particular, the results showed that the "Unary matching" technique is far better than the other two techniques and more scalable. In addition, they showed that both proposed techniques significantly outperform the state-of-the-art technique in terms of matchmaking time.

2.4.6 Conclusions

This work has proposed two novel techniques that are able to optimize the overall non-functional service matchmaking time with respect to the state-of-the-art research work. The "Unary matching" proposed technique scales better than the other one and exhibits significant advantages in matchmaking as well as insertion, deletion, and update time against the state-of-the-art work and the other proposed technique. However, it is only able to deal with unary-constrained non-functional service specifications. This disadvantage is solved by the "Subsumes matching" technique which optimizes the non-functional service matchmaking time but pays the price of higher insertion, deletion, and update time. Both techniques can be distributed and incorporated in scalable, distributed service discovery mechanisms. Future work will concentrate on not only optimizing the insertion, deletion, and update time of the "Subsumes matching"

technique but also on developing scalable and distributed, functional and non-functional service discovery mechanisms that incorporate these two novel non-functional service matchmaking techniques. In this way, if the latter mechanisms are exploited by a service broker, then the vision of the Internet of Services will come closer to its realization.

2.5 Cost Reduction Through SLA-driven Self-Management

Contributing partners: INRIA

Status: Presented at the 9th IEEE European Conference on Web Services (ECOWS'11), 14th–16th September, 2011, Lugano, Switzerland.

2.5.1 Background

The Service-Oriented Computing (SOC) promotes the conception of service-based applications on top of loosely-coupled services from distinct providers [54]. The relationship between services are defined by means of electronic contracts which are often represented by Service-Level Agreements (SLAs). The SLA defines the obligations of both provider and customer services which includes the quality that should be provided by the provider [4, 6].

Because service execution environment is distributed, service providers often take advantage of distributed computing infrastructures as clouds [1, 53, 44] and grids [17, 24]. On the one hand, clouds provide resources on-demand along with an accounting model which allows service providers to pay for resources according to their usage. On the other hand, grids provide further programming abstractions useful for managing service instances on distributed resources.

2.5.2 Problem Statement

Cost reduction is a common concern among service providers once it positively contributes to increase their profit. However, it is a challenge to reduce costs while maintaining conformance to SLAs on top of distributed infrastructures. Indeed, just assuring the quality properties described in the SLA is a complex task owing to service load fluctuations and unpredictable faults. Moreover, it is not trivial by far to understand and translate high-level quality metrics and map them to system configuration in order to properly configure service instance to meet the agreed QoS (Quality of Service). In addition to these issues, the fact of dealing with a distributed environment makes harder to build a solution for the stated problem.

2.5.3 Contribution Relevance

Most of current work which tackles SLA management in SOC does not specify actual low-level mechanisms which ensure QoS properties. These approaches typically ensure QoS by replacing services by other services which probably are more suitable for guaranteeing the QoS. Moreover, this service replacement solution is placed in the service composition level where composite services are composed based on simpler services [23, 19]. However, such approaches do not address how basic, atomic services guarantee QoS properties. Additionally, further approaches have addressed SLA management in the context of large-scale distributed applications, such as e-science applications deployed on grids, or multi-tier enterprise applications deployed on clusters [18, 5, 25]. Even though these latter approaches considers SLA aspects, they do not address meeting the business objectives of service providers which involve profit aspects.

2.5.4 Contribution Summary

This work relies on the Qu4DS (Quality Assurance for Distributed Services) framework for managing SLA by minimizing provider costs. In this context, costs refer to infrastructure usage and fine payments owing to SLA violations. In order to reduce costs on infrastructure usage, Qu4DS shares a pool of booked resources among distinct SLA contracts. Regarding fine payments, Qu4DS prevents SLA violations by providing QoS assurance mechanisms which handle dynamic events, such as resource shortages and execution faults. The QoS assurance mechanisms are guided by configurable strategies which attempt to minimize fine payments by choosing the most suitable request to abort in case of resource shortages.

Furthermore, Qu4DS integrates a rich set of QoS management mechanisms which address SLA lifecycle, i.e., from SLA template creation to service termination. Qu4DS builds on a simple interface which is compatible with modern grid and cloud IaaS interfaces. In order to map resource-level configurations to QoS metrics, Qu4DS specifically translates QoS metrics to the right resource requirements able to meet the agreed QoS. Based on the translate resource requirements, resources are acquired on-demand through a common cloud IaaS interface. Then, service instances and requests are dealt with by managing jobs on the booked resources through a grid interface which is based on the Simple Grid API (SAGA) [17]. Moreover, in order to accommodate fluctuating service loads and unpredictable faults, Qu4DS uses dynamic adaptation techniques based on multiple interacting control loops. These control loops are configurable with distinct adaptation policies which allows to extend the applicability of the framework.

2.5.5 Contribution Evaluation

The *flac2ogg* service provider was implemented by using Qu4DS in order to evaluated Qu4DS effectiveness in performing SLA-driven self-management. The flac2ogg is a service provider which encodes audio files by compressing FLAC [70] files to the OGG [71] format. It concerns a Master/Worker application which delegates to Qu4DS the task of managing the execution of its workers during request treatment. Additionally, Qu4DS also assists the flac2ogg provider by managing contract negotiation, translating QoS to resource requirements, booking resources and deploying flac2ogg instances with the right resource configuration. In addition, Qu4DS treats customer requests by reacting to resource shortages or job faults and delays which may compromise to satisfy the agreed QoS.

The Qu4DS framework has been evaluated through experiments on top of Grid5000 [9]. A total of forty-three resources were used for a customer demand composed by fifteen customers with distinct types of SLA. Qu4DS showed to be efficient by reacting to faults and thus successfully treating customer requests. During resource shortages, Qu4DS managed to choose the more suitable request to treat by aiming at increasing the provider profit. These actions were favorable for increasing the provider thus approximating the general provider profit to the ideal profit. More details about this experiment can be found in [15].

2.5.6 Conclusions

This work presented the Qu4DS framework which supports to build services on top of distributed infrastructures, such as IaaS clouds. The framework provides SLA management features which enables service provider to negotiate, instantiate and deliver their service in accordance to quality properties held by the SLA terms. More specifically, Qu4DS offer an automatic support for service execution management by taking into account SLA prices, fines, and infrastructure costs. Therefore, the approach proposed by this work fills the gap between higher-level service objectives and the runtime environment by providing actual mechanisms which manage service execution on distributed infrastructures.

Finally, Qu4DS design relies on configurable and extensible control loops which allow to increase the framework applicability to further application domains, workload characteristics and adaptation objectives. A case study was implemented and evaluates on Grid5000 which demonstrates the framework

effectiveness in increasing the provider profit while maintaining SLA compliance in dynamic and distributed environments.

2.6 Autonomic SLA-aware Service Virtualization for Distributed Systems

Contributing partners: SZTAKI, Vienna University of Technology (TUW)

Status: Presented at the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'11)

Keywords: Cloud Computing, SLA-negotiation, Service Brokering, On-demand deployment

2.6.1 Background

Cloud Computing citeBuyya2009 builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Both Grids and Service Based Applications (SBAs) already provide solutions for executing complex user tasks, but they are still lacking non-functional guarantees. The newly emerging demands of users and researchers call for expanding service models with business-oriented utilization (agreement handling) and support for human-provided and computation-intensive services. Providing guarantees in the form of Service Level Agreements (SLAs) are highly studied in Grid Computing. Nevertheless in Clouds, infrastructures are also represented as a service that are not only used but also installed, deployed or replicated with the help of virtualization.

2.6.2 Problem Statement

In Cloud infrastructures services are not only used but also installed, deployed or replicated with the help of virtualization. These services can also appear in complex business processes, which further complicates the fulfillment of SLAs in Clouds. For example, due to changing components, workload and external conditions, hardware and software failures, already established SLAs may be violated. Frequent user interactions with the system during SLA negotiation and service executions (which are usually necessary in case of failures), might turn out to be an obstacle for the success of Cloud Computing. Thus, there is demand for the development of SLA-aware Cloud middleware, and application of appropriate strategies for autonomic SLA attainment. Despite cloud computing's business-orientation, the applicability of Service-level agreements in the Cloud middleware is rarely studied, yet [76]. Most of the existing work address provision of SLA guarantees to the consumer and not necessarily the SLA-based management of loosely coupled Cloud infrastructure. In such systems it is hard to localize where the failures have happen exactly, what the reason is for the failure and which reaction should be taken to solve the problem. Such systems are implemented in a proprietary way, making it almost impossible to exchange the components (e.g. use another version of the broker). Autonomic Computing is one of the candidate technologies for the implementation of SLA attainment strategies. Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [29]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems (e.g. human body) autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware and software failures.

2.6.3 Contribution Relevance

In this contribution we have introduced a novel architecture considering resource provision using a virtualization approach and combining it with the business-oriented utilization used for the SLA agreement

handling. The solution can be used to autonomously manage diverse service infrastructures for on-demand service provision based on SLAs. We also exemplified how autonomic behaviour appears in the architecture in order to cope with changing user requirements and on demand failure handling.

2.6.4 Contribution Summary

The main contributions of this paper include: (i) the presentation of the novel loosely coupled architecture for the *SLA-based Service virtualization* and on-demand resource provision, (ii) description of the architecture including *meta-negotiation, meta-brokering, brokering and automatic service deployment* with respect to its autonomic behaviour, and (iii) the *validation* of the SSV architecture with a biochemical case study in a Cloud simulation environment.

2.6.5 Contribution Evaluation

In order to evaluate our proposed SSV solution, we use a typical biochemical application as a case study called TINKER Conformer Generator application using molecular modeling for drug development, which was gridified and tested on production Grids. The application generates conformers by unconstrained molecular dynamics at high temperature to overcome conformational bias then finishes each conformer by simulated annealing and energy minimization to obtain reliable structures. Its aim is to obtain conformation ensembles to be evaluated by multivariate statistical modeling. This use case can be regarded as a special, corresponding case of the S-Cube EHEALTH-BG-03 scenario called "Easier Planning of Examinations and Treatments". For the evaluation, we have created a general simulation environment, in which all stages of service execution in the SSV architecture can be simulated and coordinated in both Cloud and Grid environments. SLA parameters have been predefined for each task of the application, and have been evaluated resource selection at runtime. From the achieved results we can clearly see that the simulated SSV architecture using different distributed environments overperforms the former solution using only Grid resources. Comparing the different deployment strategies we can see that on demand deployment introduces some overhead, but service duplication can enhance the performance and help to avoid SLA violations with additional virtual machine deployment costs.

2.6.6 Conclusions

The presented general, conceptual SSV architecture is built on three main components: the Meta-Negotiator responsible for agreement negotiations, the Meta-Broker for selecting the proper execution environment, and the Automatic Service Deployer for service virtualization and on-demand deployment. We have also discussed how the principles of autonomic computing are incorporated to the SSV architecture to cope with the error-prone virtualization environments. The proposed service virtualization architecture is validated in a simulation environment based on CloudSim, using a general biochemical application as a case study. The evaluation results clearly fulfil the expected utilization gains compared to a less heterogeneous Grid solution.

Chapter 3

Conclusions

3.1 Outlook and Future Research Challenges

This deliverable introduces a novel service runtime infrastructure, which will incorporate an active and QoS-aware registry and client components. This infrastructure ensures SLA compliance and suggests services as well as ad-hoc processes. The presented results related to elements and aspects of infrastructure and higher level mechanisms for specifying SLAs in regard of QoS requirements. These mechanisms are targeting the objectives of the 2.3 package.

To summarize the significant contributions: TUW studied the combination of service metadata, Quality of Service, service querying, dynamic binding and service mediation. Further, in their work the cost of adaptation on SLA violations is discussed. A third aspect is the establishment of some QoS in a service-based crowd environment. UoC propose two alternative techniques for improving the non-functional service matchmaking time. INRIA's studies consider a SLAs management which satisfies the customers requirements and also their own business objectives, such as maximizing profits. Finally, SZTAKI introduces an SLA-aware Service Virtualization architecture that provides non-functional guarantees in the form of SLAs and consists of a three-layered infrastructure including agreement negotiation, service brokering and on demand deployment.

The deliverable is a collection of scientific papers, either published in conference proceedings, journals, or very recent work still under review, and organized along the research directions of WP-JRA-2.3. The papers all have been peer reviewed which ensures that the papers represent significant contributions to service-based system research and they demonstrate a final progress in the WP. The positioning of the papers within the adaptation framework, their relationship to the WP-JRA-2.3 research goals and vision and to other research WPs are exposed in Section 1.5. A more in detail discussion follows in Chapter 2. Here each contribution gives background information, states the problem statement, and the contribution relevance. This is followed by a contribution summary, an evaluation description and a conclusion with future ideas.

Despite the fact that the present deliverable is the last in the JRA-2.3 Workpackage series, all contributors agree, that future work will not only continue on individual tracks, but also consider collaborations that have certainly also been promoted in the context of SCube. The work on Self-* Service Infrastructures and Service Discovery Support will continue also after the project's end with the hot topic of Cloud Computing rising in the service infrastructure community. The trend can also be recognized from some of the contributions in this deliverable.

Bibliography

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, April 2011.

[2] Amazon Mechnical Turk. http://www.mturk.com, last access March 2011.

[3] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Open Grid Forum (OGF), 2006. http://www.gridforum.org/documents/GFD.107.pdf, Last Visited: 2011-07-19.

[4] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, 2007.

[5] Siegfried Benkner and Gerhard Engelbrecht. A Generic QoS Infrastructure for Grid Web Services. *Advanced International Conference on Telecommunications / Internet and Web Applications and Services, International Conference on*, 0:141, 2006.

[6] Philip Bianco, Grace A. Lewis, and Paulo Merson. Service Level Agreements in Service-Oriented Architecture Environments. Technical Report CMU/SEI-2008-TN-021, Software Engineering Institute of The Carnegie Mellon University, http://www.sei.cmu.edu/reports/08tn021.pdf, 2008.

[7] David Bodoff, Mordechai Ben-Menachem, and Patrick C.K. Hung. Web Metadata Standards: Observations and Prescriptions. *IEEE Software*, 22(1):78–85, 2005.

[8] D.C. Brabham. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75, 2008.

[9] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Owen Cliffe and Dimitris Andreou. Service Matchmaking Framework. Public Deliverable D5.2a, Alive EU Project Consortium, 10 September 2009. Available at: http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=28&Itemid=49.

[11] Antonio Ruiz Cortés, Octavio Martín-Díaz, Amador Durán Toro, and Miguel Toro. Improving the Automatic Procurement of Web Services Using Constraint Programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005.

[12] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard P. King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer, and Alaa Youssef. Web Services on Demand: WSLA-Driven Automated Management. *IBM Systems Journal*, 43:136–158, January 2004.

[13] Seema Degwekar, Stanley Y. W. Su, and Herman Lam. Constraint Specification and Processing in Web Services Publication and Discovery. In *ICWS*, pages 210–217, 2004.

[14] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Mass collaboration systems on the World Wide Web. *Communications of the ACM*. to appear.

[15] André Lage Freitas, Nikos Parlavantzas, and Jean-Louis Pazat. Cost Reduction Through SLA-driven Self-Management. In *In Proceedings of The 9th IEEE European Conference on Web Services (ECOWS'11)*, September 2011.

[16] Andreas Friesen and Michael Altenhofen. Matching Composed Semantic Web Services at Publishing Time. In *Proceedings of the IEEE International Conference on E-Commerce Technology Workshops*, pages 64–70, Munich,Germany, 2005. IEEE Computer Society.

[17] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. Saga: A simple api for grid applications - high-level application programming on the grid. *Computational Methods in Science and Technology: special issue "Grid Applications: New Challenges for Computational Methods"*, SC05:8(2), November 2005.

[18] Peer Hasselmeyer, Bastian Koller, Lutz Schubert, and Philipp Wieder. Towards SLA-Supported Resource Management. In *HPCC '06: Proceedings of the 2006 International Conference on High Performance Computing and Communications*, pages 743–752. Springer, 2006.

[19] Julia Hielscher, Andreas Metzger, and Raman Kazhamiakin. Taxonomy of Adaptation Principles and Mechanisms. Technical Report Deliverable # CD-JRA-1.2.2, S-CUBE Consortium, 2009.

[20] Waldemar Hummer, Philipp Leitner, and Schahram Dustdar. SEPL – A Domain-Specific Language and Execution Environment for Protocols of Stateful Web Services. *Distributed and Parallel Databases*, 29:277–307, August 2011.

[21] C. Hwang and K. Yoon. Multiple Criteria Decision Making. *Lecture Notes in Economics and Mathematical Systems*, 1981.

[22] Panagiotis G. Ipeirotis. Analyzing the Amazon Mechanical Turk Marketplace. *SSRN eLibrary*, 17(2):16–21, 2010.

[23] Florian Irmert, Thomas Fischer, and Klaus Meyer-Wegener. Runtime adaptation in a service-oriented component model. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 97–104, New York, NY, USA, 2008. ACM.

[24] Shantenu Jha, Andre Merzky, and Geoffrey Fox. Using Clouds to Provide Grids with Higher Levels of Abstraction and Explicit Support for Usage Modes. *Concurrency and Computation: Practice & Experience*, 21:1087–1108, 2009.

[25] Jose Antonio Parejo and Pablo Fernandez and Antonio Ruiz-Cortés and José María García. SLAWs: Towards a Conceptual Architecture for SLA Enforcement. In *Services, IEEE Congress on*, volume 0, pages 322–328. IEEE Computer Society, 2008.

[26] Lukasz Juszczyk, Hong-Linh Truong, and Schahram Dustdar. GENESIS - A Framework for Automatic Generation and Steering of Testbeds of Complex Web Services. In *Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'08)*. IEEE Computer Society, 2008.

[27] Raman Kazhamiakin, Branimir Wetzstein, Dimka Karastoyanova, Marco Pistore, and Frank Leymann. Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In *Proceedings of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, pages 395–404, 2009.

[28] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal on Network and Systems Management*, 11:57–81, March 2003.

[29] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[30] A. Kertész, G. Kecskemeti, and I. Brandic. Autonomic sla-aware service virtualization for distributed systems. In *In proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, 2011.

[31] Matthias Klusch, Benedikt Fries, and Katia Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):121 – 133, 2009.

[32] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11:60–67, November 2007.

[33] K. Kritikos and B. Pernici. Initial concepts for specifying end-to-end quality characteristics and negotiating slas. Technical report, 2009.

[34] Kyriakos Kritikos. QoS-based Web Service Description and Discovery. PhD Thesis, Computer Science Department, University of Crete, Heraklion, Greece, December 2008.

[35] Kyriakos Kritikos, Fabio Paternò, and Dimitris Plexousakis. Towards Identifying Services to Realize the Functionality of Interactive Applications based on User Task Models. *ACM Transactions on Interactive Intelligent Systems*, 2011. under review.

[36] Kyriakos Kritikos and Dimitris Plexousakis. Mixed-Integer Programming for QoS-Based Web Service Matchmaking. *IEEE Trans. Serv. Comput.*, 2(2):122–139, 2009.

[37] André Lage Freitas, Nikos Parlavantzas, and Jean-Louis Pazat. Cost Reduction Through SLA-driven Self-Management. In *European Conference on Web Services (ECOWS)*, Lugano, Switzerland, September 2011. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant greement 215483 (S-CUBE). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

[38] P. Leitner. Requirements for service infrastructures in dynamic environments and evaluation of existing service registries. Technical report, 2009.

[39] P Leitner, B Wetzstein, F Rosenberg, A Michlmayr, S Dustdar, and F Leymann. Runtime Prediction of Service Level Agreement Violations for Composite Services. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, pages 176–186, Berlin, Heidelberg, 2009. Springer-Verlag.

[40] Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. Cost-Based Optimization of Service Compositions. *IEEE Transactions on Services Computing (TSC)*, 2011. To appear.

[41] Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Daios – Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13(3):30–38, 2009.

[42] Philipp Leitner, Branimir Wetzstein, Dimka Karastoyanova, Waldemar Hummer, Schahram Dustdar, and Frank Leymann. Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'10)*. Springer, 2010.

[43] Tammo van Lessen, Jörg Nitzsche, and Frank Leymann. Formalising Message Exchange Patterns using BPEL Light. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08)*, pages 353–360, Washington, DC, USA, 2008. IEEE Computer Society.

[44] Lillard, Terrence V. and Garrison, Clint P. and Schiller, Craig A. and Steele, James. *The Future of Cloud Computing*, pages 319–339. Elsevier, 2010.

[45] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73, 2004.

[46] Juval Löwy. *Programming WCF Services*. O'Reilly, 2007.

[47] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2), 2001.

[48] Daniel A. Menascé. QoS Issues in Web Services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[49] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Service Provenance in QoS-Aware Web Service Runtimes. In *Proceedings of the 7th International Conference on Web Services (ICWS'09)*. IEEE Computer Society, 2009.

[50] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Transactions on Services Computing*, 3:193–205, July 2010.

[51] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective. In *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07), co-located with ESEC/FSE'07*. ACM, 2007.

[52] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valérie Issarny, and Yolande Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *J. Syst. Softw.*, 81(5):785–808, 2008.

[53] Nurmi, Daniel and Wolski, Rich and Grzegorczyk, Chris and Obertelli, Graziano and Soman, Sunil and Youseff, Lamia and Zagorodnov, Dmitrii. The Eucalyptus Open-Source Cloud-Computing System. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[54] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40:38–45, 2007.

[55] M. Parkin and A. Metzger. Initial set of principles, techniques and methodologies for assuring end-to-end quality and monitoring of slas. Technical report, 2010.

[56] Pierluigi Plebani and Barbara Pernici. URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642, 2009.

[57] H. Psaier, L. Juszczyk, F. Skopik, D. Schall, and S. Dustdar. Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems. In *SASO*, pages 164–174. IEEE, 2010.

[58] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[59] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An End-to-End Approach for QoS-Aware Service Composition. In *Proceedings of the 13th International Enterprise Computing Conference (EDOC'09)*. IEEE Computer Society, 2009.

[60] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, and Schahram Dustdar. Integrated Metadata Support for Web Service Runtimes. In *Proceedings of the Middleware for Web Services Workshop (MWS'08)*, pages 361–368, Washington, DC, USA, 2008. IEEE Computer Society.

[61] Florian Rosenberg, Christian Platzer, and Schahram Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 205–212, Washington, DC, USA, 2006. IEEE Computer Society.

[62] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[63] O. Sammodi and A. Metzger. Integrated principles, techniques and methodologies for specifying end-to-end quality and negotiation slas and for assuring end –to-end quality provision and sla conformance (incl. proactiveness). Technical report, 2011.

[64] B. Satzger, H. Psaier, D. Schall, and S. Dustdar. Stimulating skill evolution in market-based crowdsourcing. In *9th International Conference on Business Process Management (BPM)*, volume 6896 of *Lecture Notes in Computer Science*, pages 66–82. Springer, 2011.

[65] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley, New York, N.Y, USA, 1986.

[66] F. Silvestri. Knowledge extraction of service usage. Technical report, 2010.

[67] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise Service Level Agreements. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society.

[68] Florian Skopik, Daniel Schall, and Schahram Dustdar. Trust-Based Adaptation in Complex Service-Oriented Systems. In *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 31–40, Washington, DC, USA, 2010. IEEE Computer Society.

[69] M. Stollberg, M. Hepp, and J. Hoffmann. A Caching Mechanism for Semantic Web Service Discovery. In *ICWS*, 2007.

[70] The FLAC project. Free Lossless Audio Codec (FLAC). http://flac.sourceforge.net/, 2011.

[71] The Xith Open Source Community. Ogg Vorbis Audio Format. http://www.vorbis.com/, October 2011.

[72] M. Vukovic. Crowdsourcing for Enterprises. In *Proceedings of the 2009 Congress on Services*, pages 686–692. IEEE Computer Society, 2009.

[73] Yao Wang and Julita Vassileva. Toward Trust and Reputation Based Web Service Selection: A Survey. *International Transactions on Systems Science and Applications (ITSSA)*, 3(2), 2007.

[74] Branimir Wetzstein, Philipp Leitner, Florian Rosenberg, Schahram Dustdar, and Frank Leymann. Identifying Influential Factors of Business Process Performance Using Dependency Analysis. *Enterprise Information Systems*, 4(3):1–8, July 2010.

[75] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0 Part 0: Primer - W3C Candidate Recommendation 27 March 2006, 2006. http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/, Last Visited: 2011-07-19.

[76] C. A. Yfoulis and A. Gounaris. Honoring slas on cloud computing services: a control perspective. In *Proceedings of the European Control Conference*, 2009.

[77] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.

[78] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee. DAML-QoS Ontology for Web Services. In *ICWS*, page 472, San Diego, CA, USA, 2004. IEEE Computer Society.

Appendix A

Attached Papers

Stimulating Skill Evolution in Market-based Crowdsourcing

Benjamin Satzger, Harald Psaier, Daniel Schall, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
{satzger,psaier,schall,dustdar}@infosys.tuwien.ac.at,
WWW home page: http://www.infosys.tuwien.ac.at

Abstract. Crowdsourcing has emerged as an important paradigm in human problem-solving techniques on the Web. One application of crowdsourcing is to outsource certain tasks to the crowd that are difficult to implement in software. Another potential benefit of crowdsourcing is the on-demand allocation of a flexible workforce. Businesses may outsource tasks to the crowd based on temporary workload variations. A major challenge in crowdsourcing is to guarantee high-quality processing of tasks. We present a novel crowdsourcing marketplace that matches tasks to suitable workers based on auctions. The key to ensuring high quality lies in skilled members whose capabilities can be estimated correctly. We present a novel auction mechanism for skill evolution that helps to correctly estimate workers and to evolve skills that are needed. Evaluations show that this leads to improved crowdsourcing.

Keywords: Human-centric BPM, Crowdsourcing, Online communities, Task markets, Auctions, Skill evolution

1 Introduction

Today, ever changing requirements force in-house business processes to rapidly adapt to changing situations in order to stay competitive. Changes involve not only the need for process adaptation, but also, require an additional inclusion of new capabilities and knowledge, previously unavailable to the company. Thus, outsourcing of parts of business processes became an attractive model. This work, in particular, focuses on a distinguished recent type of outsourcing called *crowdsourcing*. The term crowdsourcing describes a new web-based business model that harnesses the creative solutions of a distributed network of individuals [4], [18]. This network of humans is typically an open Internet-based platform that follows the *open world* assumption and tries to attract members with different knowledge and interests. Large IT companies such as Amazon, Google, or Yahoo! have recognized the opportunities behind such *mass collaboration systems* [8] for both improving their own services and as business case. In particular, Amazon focuses on a task-based marketplace that requires explicit collaboration. The most prominent platform they currently offer is *Amazon Mechanical Turk* (AMT) [3].

Requesters are invited to issue *human-intelligence tasks* (HITs) requiring a certain qualification to the AMT. The registered customers post mostly tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [11]).

Reasons for companies to outsource tasks to a crowd platform are flexibility, high availability, and low costs. Workers from all over the world and different timezones are allowed to register with the platform. With 90% of tasks processed at a cost of $0.10 and less, the same task is usually also offered to multiple AMT workers. However, crowdsourcing platforms are loosely-coupled networks with members of different interests, working style, and cultural background. The flexible crowd structure allows workers to join and leave at any time. This hampers to predict or guarantee the quality of a task's result. There are crowd providers such as CrowdFlower [6] that broker crowd resources to customers to overcome quality and reliability issues.

However, managing and adapting the crowd's skills and resources in an automated manner remains challenging. Crowd customers prefer fully automated deployment of their tasks to a crowd, just as in common business process models. In this paper, we base our solution on the service-oriented architecture (SOA) paradigm. SOAs are an ideal grounding for distributed environments. With their notion of the participants as services and registries, resources can be easily and even automatically discovered for composing whole business processes. A plethora of standards supports seamless integration and registration of new services, and provides protocols for communication, interaction and control of the components. Altogether, we believe SOAs provide an intuitive and convenient technical grounding to automate large-scale crowdsourcing environments. Important to note, today SOA not only includes software-based services, but also *Human-Provided Services* [16] and *BPEL4People* [2] for human interactions in business processes and allow to host mass collaboration environments.

The main challenges addressed in this work relate to building and managing an automated crowd platform. It is not only of importance to find suitable workers for a task and to provide the customer with satisfying quality, but also, to maintain a motivated base of crowd members and provide stimulus for learning required skills. Only a recurring, satisfied crowd staff is able to ensure high quality and high output. As any crowd, fluctuations must be compensated and a *skill evolution* model must support new and existing crowd workers in developing their capabilities and knowledge. Finally, the standard processes on such a platform should be automated and free from intervention to handle the vast amount of tasks and to make it compatible with a SOA approach. Atop, the model should increase the benefit of all participants. In detail, we provide the following contributions in this paper:

- *Automated matching and auctions.* For providing a beneficial distribution of the tasks to the available resources we organize auctions according to novel mechanisms.
- *Stimulating skill evolution.* In order to bootstrap new skills and unexperienced workers we provide skill evolution by integrating assessment tasks into our auction model.

– *Evaluation.* Experiments quantify the advantages of a skill evolution based approach in comparison to traditional auctions.

The remainder of this paper is structured as follows: In Section 2 related work is discussed. Section 3 describes the design of our crowdsourcing system, including its actors and their interaction. Then, Section 4 details our adaptive auction mechanisms and Section 5 presents the conducted experiments and discusses their results. Section 6 concludes the paper and points to future work.

2 Related Work

In this work we position crowdsourcing in a service-oriented business setting by providing automation. In crowdsourcing environments, people offer their skills and capabilities in a service-oriented manner. Major industry players have been working towards standardized protocols and languages for interfacing with people in SOA. Specifications such as WS-HumanTask [10] and BPEL4People [2] have been defined to address the lack of human interactions in service-oriented businesses [14]. These standards, however, have been designed to model interactions in closed enterprise environments where people have predefined, mostly static, roles and responsibilities. Here we address the service-oriented integration of human capabilities situated in a much more dynamic environment where the availability of people is under constant flux and change [5]. The recent trend towards *collective intelligence* and crowdsourcing can be observed by looking at the success of various Web-based platforms that have attracted a huge number of users. Well known representatives of crowdsourcing platforms include Yahoo! Answers [20] (YA) and the aforementioned AMT [3]. The difference between these platforms lies in how the labor of the crowd is used.

YA, for instance, is mainly based on interactions between members. Questions are asked and answered by humans, thereby lacking the ability to automatically control the execution of tasks. The interesting aspect of YA relevant for our crowdsourcing approach is the role of *two-sided markets* [13]. In YA, users get 100 points by signing-up to the platform [19]. For each answer being provided, users get additional points (more points if the answer is selected as best answer). However, users get negative points if they ask questions, thereby encouraging members to provide answers. Based on the rewarding scheme in YA, users tend to have either role – being answerer or asker – instead of having both roles. In the context of YA and human-reviewed data, [17] provided an analysis of data quality, throughput and user behavior. In contrast, AMT offers access to the largest number of crowdsourcing workers. With their notion of HITs that can be created using a Web service-based interface they are closely related to our aim of mediating the capabilities of crowds to service-oriented business environments. According to one of the latest analysis of AMT [11], HIT topics include, first of all, transcription, classification, and categorizations tasks for documents and images. Furthermore, there is also tasks for collecting data, image tagging, and feedback or advice on different subjects. The major challenges are to find

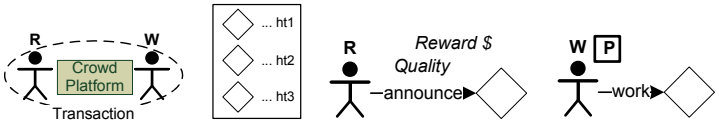
on request skilled workers that are able to provide high quality results for a particular topic (e.g., see [1]), to avoid spamming, and to recognize low-performers. To the best of our knowledge, these problems are still not faced by AMT. In this work we focus on those issues. The shortcoming of most existing real platforms is the lack of detailed *skill information*. Most platforms have a simple measure to prevent workers (in AMT, a threshold of task success rate can be defined) from claiming tasks. In [15], the automated calculation of expertise profiles and skills based on interactions in collaborative networks was discussed. In this work, we introduce a novel auction mechanism to promote skill evolution in crowdsourcing environments. Compared to open bidding systems used by popular online auction markets such as eBay [9], our auction mechanism is a closed bidding system similar to public procurement. In [7], a model of crowdsourcing is analyzed in which strategic players select among, and subsequently compete in, contests. In contrast, the presented approach in this paper supports the evolution of worker skills through auction mechanisms.

3 Design of Marketplaces in Crowdsourcing

The core activity in task-based crowd environments is members providing their labor by processing tasks. In this section, we explain our idea of task-based crowdsourcing on a market-oriented platform. The aim is to organize and manage the platform to the satisfaction and benefit of all participants; crowd members and platform provider. We will now introduce the basic design of the proposed crowdsourcing environment.

3.1 Skill-based Task Markets

In task markets different stakeholders can be identified. Generally, there is the requesters and workers representing the registered members of a crowd marketplace. The task of the third stakeholder, the crowd operator in between, is to manage the crowd task auctions. To satisfy any of the stakeholders the operator must assure that the requesters obtain a result of high quality in a timely manner. On the other hand, the workers would like to have tasks available whenever they are motivated to work and are interested in a high reward for processing a task. The operator itself works towards a long-term profit. To bootstrap the skill-based system, each member interested in offering of processing tasks is required to create a profile containing information about her/his interests and skills. The basic interactions and an external view on the proposed crowdsourcing environment are depicted in Fig. 1. The crowdsourcing environment consists of members who can participate in transactions (see Fig. 1(a)). Within a particular transaction a member can either adopt the role of a requester **R**, who initiates a transaction by announcing tasks (see Fig. 1(b)), or the role of a worker **W**, who processes a task. We propose a crowdsourcing marketplace that handles transactions transparently for its members; requesters and workers do not communicate directly, but only with the crowdsourcing marketplace in between. We



(a) Crowd. (b) Tasks. (c) Requester. (d) Worker.

Fig. 1. Crowd environment building blocks and interaction of stakeholders.

argue that this standardized style of interaction is less prone for misconceptions and more efficient because it allows members getting used to the system. Tasks (Fig. 1(b)) are created by requesters based on their current needs. Requesters initiate a transaction by submitting a task to the marketplace, with additional information about the amount of money he is willing to pay for the processing of the task and additional requirements (Fig. 1(c)). It is the responsibility of the marketplace operator to find a suitable worker, to submit the task to the worker, to collect the result, and to transmit it to the requester.

The interaction of a worker with the market platform is initiated by the latter by asking a member whether s/he is interested in processing a task (Fig. 1(d)). This interest can be expressed by bidding for the task. Workers have skill profiles denoted by the symbol **P**. These profiles are not statically defined, but are updated based on the level of delivered task quality. This procedure ensures an up-to-date view on workers' capabilities and skills. Based on the bids and background information about the bidders, the system selects one or multiple workers, who are then asked to process the task.

3.2 Towards Auction-based Crowdsourcing

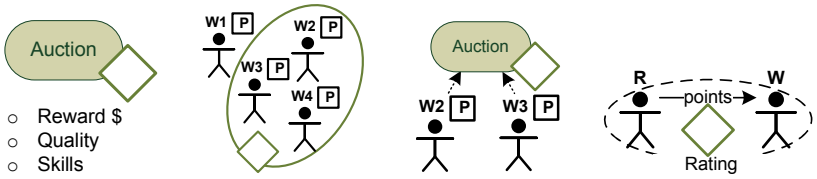
Auctions are a very old idea already used by the Babylonians but still an active area of research. The rise of e-commerce has drastically increased the number and diversity of goods traded via auctions. Many recently installed markets, such as energy or pollution permit markets, are based on auctions [12]. There are many different flavors of auctions differing in the number of items considered (single/multi item), the number of buyers and sellers (demand/supply/double auction), the bidding procedure (open/closed bids and ascending/descending), and how the price is determined (e.g., first/second price); however, four standard types are widely used [12]. They all assume a single seller and multiple buyers (demand auction) and, in their simplest forms, a single item to sell (single-item auction). The so-called English auction is an ascending open-bid auction, the Dutch auction is a descending open-bid auction. The other two standard auction types are closed-bid auctions, i.e., each bidder submits a single bid which is hidden for other buyers. We use an adapted version of a closed-bid auction; a single auction deals with the matching of one task to one or many crowd workers (single-item demand auction). We will introduce our crowdsourcing auction mechanisms in the following sections.

4 Auction-based Task Assignment

In the following we discuss the steps involved in transaction processing and outline the novel idea of skill evolution.

4.1 Processing of Transactions

Figure 2 illustrates the steps involved in the internal processing of a transaction. In the qualification step the marketplace identifies all members capable of processing the task (Fig. 2(a)), based on the task description and the members' profiles. The preselection chooses a limited number of the most suitable workers (Fig. 2(b)) to have a reasonable amount of participants for the auction. The preselection step helps to avoid a flooding of auction announcements. In this way members are only exposed to tasks/auctions for which they actually have a realistic chance of being accepted as worker. Due to the voluntary, open nature of crowdsourcing environments, not all preselected workers may decide to follow the invitation to compete in an auction.



(a) Auction. (b) Preselection. (c) Biddings. (d) Feedback.

Fig. 2. Internal processing of a transaction.

This fact is depicted by the transition phase between Fig. 2(b) and Fig. 2(c) where only a subset of preselected workers decides to participate. The auction phase (Fig. 2(c)) allows each participant to submit an offer and finally, a winner is determined who is supposed to process the task. In the case of a successful processing the marketplace returns the final result to the requester, handles the payment, and allows the requester to give feedback about the transaction in the form of a rating (Fig. 2(d)).

As mentioned before, tasks come with a description, a maximum amount of money the requester is willing to pay and further requirements, i.e., time requirements and quality requirements. The former is typically given in the form of a deadline, the latter could range from a simple categorization (e.g., low, high) to sophisticated quality requirement models. Each worker has a self-provided profile describing her/his skills and, additionally, the marketplace operator is keeping track of the actual performance. We propose to maintain a performance value per user and skill, encoded as tuple consisting of the observed performance and the confidence in that value. The input used to generate these performance values comes from the ratings of the requesters and a check whether the deadline was met, which can be performed by the system without feedback from the

requester. The qualification phase is based on a matching of the task description to the skills of the members considering their performance and confidence values. Higher requirements impose higher standards on the performance of the member. The result of this matching is a boolean value indicating whether a member is meeting the minimum requirements. In the next step, the preselection, the qualified members are ranked based on skill, performance, and the confidence in the performance; only the top-k members are chosen to participate in the auction. This helps to reduce the number of auction requests to members in order to avoid spamming members and to spare members the frustration caused by not winning the auction. The marketplace operator as the auctioneer hides parts of the task's data. Workers only see the task description and the time and quality requirements, but not the associated price determined by the requester. The auction is performed as a closed bid auction, whereas each participant is only allowed one bid. At the end of the auction a winner is determined based on the amounts of the bids and the performance-confidence combination of the bidders' skills. If all bids are higher than the amount the requester is willing to pay the auctioneer would typically reject all bids and inform the requester that the task cannot be assigned under the current conditions. In this case the requester could change the task by increasing the earnings, lowering the quality requirements or extending the deadline and resubmit the task. With a selection strategy outlined in more detail in the next section the marketplace assigns the task to the worker for processing. After the processing of the task by the worker and the receipt of rating information, the performance of the worker is adjusted and the confidence value is increased.

Technically, an aptitude function estimates how well workers are suited for handling a task. It is used as basis for qualification and preselection and can be formally defined as

$$aptitude : W \times T \rightarrow [0, 1], \tag{1}$$

where W is the set of workers and T represents tasks. $aptitude(w, t) = 1$ would mean that worker $w \in W$ is perfectly qualified for handling task $t \in T$. A mapping to zero would represent a total inaptness. Similarly, a ranking function is used to rank workers' bids:

$$rank : W \times T \times B \rightarrow [0, 1], \tag{2}$$

where B is the set of bids. In addition to the aptitude, the *rank* function also takes monetary aspects, contained in bid $b \in B$, into account. A property of a sound ranking function is that if two workers have the same aptitude for a task then the one with the lower bid will have a higher rank. The *aptitude* function is used for performing qualification and preselection. As auction admittance strategy you can either admit all workers with an aptitude higher than a certain threshold, the top-k workers according to aptitude, or a combination of the two strategies. The ranking function is used to determine the winner of an auction: the highest ranked worker.

4.2 Skill Evolution

The concepts discussed so far provide the fundamentals for automated matching of tasks to workers. As outlined previously, a further major challenge hampering the establishment of a new service-oriented computing paradigm spanning enterprise and open crowdsourcing environments are quality issues. In our scenario this is strongly connected to correctly estimating the skills of workers. One approach for increasing the confidence in worker skills are qualification tasks, with the shortcoming that these tasks would need to be created (manually) by the requesters who have the necessary knowledge. This implies a huge overhead for the testing requester; s/he is also the only one who benefits from the gathered insights. Here, we take a different approach by integrating the capability of confidence management into the crowdsourcing platform itself. Instead of having point-to-point tests, we propose the automated assessment of workers to unburden requesters in inspecting workers' skills. We believe that this approach offers great potential for the (semi-)automatic inclusion of crowd capabilities in business environments. The first challenge one needs to address is to cope with the "hostile" environment in which computing is performed. Workers may cheat on results (e.g., copy and paste of existing results available in the platform), spam the platform with unusable task results, or even provide false information. A well-known principle in open, Web-based communities is the notion of *authoritative* sources that act as points of references. For example, this principle has been applied on the Web to propagate trust based on *good seeds*. Our idea of skill evolution is in a manner similar. We propose the *automatic assessment* of workers where confidence values are low. For example, newcomers who recently signed up to the platform may be high or low performers. To unveil the tendency of a worker, we create a hidden 'tandem' task assignment comprising a worker whose skills are known (high performer) with a high confidence and a worker where the crowdsourcing platform has limited knowledge about its skills (i.e., low confidence). The next step is that both workers process the *same* task in the context of a requester's (real) task. However, only the result of the high confidence worker is returned to the requester, whereas the result of the low confidence worker is compared against the delivered reference. This approach has advantages and drawbacks. First, skill evolution through tandem assignments provides an elegant solution to avoid training tasks (assessments are created automatically and managed by the platform) and also implicitly stimulates a learning effect. Of course, the crowdsourcing platform cannot charge the requester for tandem task assignments since it mainly helps the platform to better understand the true skill (confidence) of a worker. Thus, the platform must pay for worker assessments. As we shall show later in our evaluation, performing assessments provides the positive effect that the overall quality of provided results and thus requester satisfaction increases due to a better understanding of worker skills. We embed skill evolution in our crowdsourcing platform as follows. After the winner of an auction has been determined it is evaluated whether an assessment task is issued to further workers. The function *assess* outputs 1 if an assessment task is to be assigned to a worker and 0 otherwise.

$$assess : W \times T \times B \times W \rightarrow \{0, 1\} \quad (3)$$

An input tuple (w, t, b, w_r) checks whether tasks $t \in T$ is to be assigned to $w \in W$ who offered bid $b \in B$. Worker $w_r \in W$ is the reference worker, in our case the worker who has won the corresponding auction and who will thus process the same task.

5 Implementation and Evaluation

In this section we discuss implementation aspects, introduce the detailed design of our experiments and present results.

5.1 Simulation Environment

We have implemented a Java-based simulation framework that supports all previously introduced concepts and interactions between requesters, the platform, and workers. All of the above introduced functions (1)-(3) have been implemented in our framework. Due to space limitations, we do not present a detailed discussion on implementation aspects. The interested reader can find details regarding the prototype as well as a Web-based demo online[1].

5.2 Experiment Design and Results

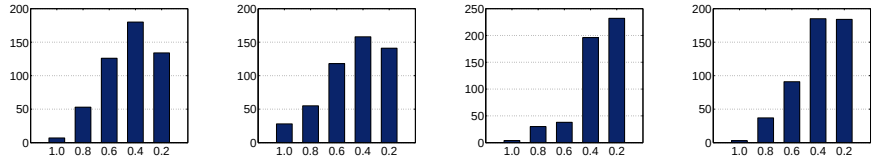
An evaluation scenario consists of a set of workers W and a set of requesters R. In every round of the simulation each requester usually announces a task. An auction is conducted for each announced task t, which consists of a description of the skills needed for its processing, an expected duration, a deadline, and the expected quality. High quality requirements indicate highly sophisticated and demanding tasks. For each worker w and skill s the platform maintains a performance value $pfmc(w, s)$ and a confidence in that value $cnfd(w, s)$. This observed performance value is derived from requester ratings; if it is based on many ratings the confidence is close to one, if there are only a few ratings available the confidence is close to zero. Based on task t's skill requirements and a worker w's performance/confidence values for these skills it is possible to calculate the expected performance $pfmc(w, t)$ and confidence $cnfd(w, t)$ for that task. For the evaluation we assume that each worker w has a certain performance $pfmc_{real}(w, s)$ for a skill s which is hidden but affects the quality of the results. Requesters rate the workers based on the results which in turn is the basis for the observed performance and confidence values. We assume that the processing of tasks demanding a certain skill causes a training effect of that skill, i.e., $pfmc_{real}(w, s)$ increases. For the sake of simplicity in the evaluation we assume that there is only one skill. This does not change fundamental system properties and one skill allows to extrapolate the behavior of multiple skills. Whether a single or multiple skills are considered indeed affects qualification, preselection, bidding, and rating but all the mechanisms are naturally extensible from one to

[1] `http://www.infosys.tuwien.ac.at/prototyp/Crowds/Markets_index.html`

multiple skills by performing the same computations for each skill and a final combination step. Hence, $pfmc : W \times S \rightarrow [0, 1]$ and $pfmc : W \times T \rightarrow [0, 1]$ are reduced to $pfmc : W \rightarrow [0, 1]$. The same holds for $cnfd$ and $pfmc_{real}$.

For the simulation we have created 500 workers with random values for $pfmc_{real}(w)$ and $cnfd(w)$ according to a normal distribution $\mathcal{N}(\mu, \sigma^2)$. The initial performance value $pfmc(w)$ is set according to the formula $pfmc_{real}(w) + \mathcal{N}(0, 1-cnfd(w))$ which ensures that for high confidence the expected deviation of $pfmc(w)$ from $pfmc_{real}(w)$ is small and for low confidence values it is high, respectively. All values are restricted to the range of $[0, 1]$. The following figures illustrate the simulation setup in detail.

Scenario 1 assumes that there are three requesters (i.e., typically three tasks are issued in every round). It is an environment in which skilled workers are rare and the confidence in the workers' performance is relatively low, i.e., there are many workers who have few ratings. The real performance $pfmc_{real}(w)$ for a worker w is drawn according to $\mathcal{N}(0.3, 0.25)$, the confidence value $cnfd(w)$ is randomly generated by $\mathcal{N}(0.2, 0.25)$. Given the two generated values $pfmc_{real}(w)$ and $cnfd(w)$ the observed performance is randomly drawn according to $\mathcal{N}(pfmc_{real}(w), 1-cnfd(w))$. Hence, a low confidence in the performance leads to highly distorted values for $pfmc(w)$, higher confidence values decrease the variance. Figure 3 gives a detailed view on the statistical distributions of the workers' performance and confidence in our experiments.

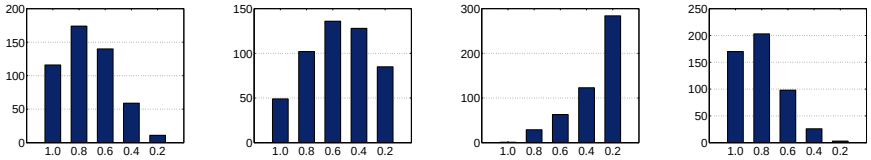


(a) real pfmc (b) obs pfmc (c) diff real/obs (d) cnfd

Fig. 3. Generated worker population according to Scenario 1.

The histrograms count the number of workers in the buckets $[0, 0.2)$, $[0.2, 0.4)$, $[0.4, 0.6)$, $[0.6, 0.8)$, and $[0.8, 1]$ according to real performance $pfmc_{real}$ in Fig. 3(a) and according to the observed performance $pfmc$ in Fig. 3(b). Fig. 3(c) represents the difference of real to observed performance; the confidence $confd$ values is shown in Fig. 3(d).

Scenario 2 contains 20 requesters which results in a much more "loaded" system. The workers are relatively skilled; their real performance $pfmc_{real}(w)$ is generated according to $\mathcal{N}(0.7, 0.25)$, i.e., the mean performance value is 0.7 compared to 0.3 as in the previous scenario. We further assume that there is a higher amount of ratings already available and $cnfd(w)$ is generated according to $\mathcal{N}(0.8, 0.25)$. Performance values are again generated as $\mathcal{N}(pfmc_{real}(w), 1-cnfd(w))$. The figures of Fig. 4 illustrate the generated worker population for Scenario 2 in the same way as for the previous one. In both scenarios tasks are issued in the first 500 rounds and the simulation ends when all workers have finished all accepted tasks.



(a) real pfmc (b) obs pfmc (c) diff obs/real (d) cnfd

Fig. 4. Generated worker population according to Scenario 2.

Tasks are generated randomly with the following methodology. The real hidden result of a task is set to a uniformly distributed random value $\mathcal{U}(0,1)$ from the range $[0,1]$. An expected duration is randomly drawn from the set $\{1,2,\ldots,24\}$. A randomly assigned quality requirement $\mathcal{U}(0.4,1)$ states whether the task poses high requirements to its processing. This means that requesters never state that the quality of their tasks' results is allowed to be lower than 0.4. Last but not least a random deadline is generated for which is guaranteed that it is after the expected duration.

Requester Behavior. In every round of the simulation each requester is asked to submit a task. After processing requesters receive the result for the task. If they receive the result after the deadline they rate the transaction with a value of zero and suspend for 20 rounds, i.e., they would refuse to issue a new task due to the negative experience. If the result is transmitted on time requesters rate the quality of the received result. Computationally this is done by comparing the task's real result with the received result. It is assumed that task requesters are able to estimate whether the received result is close to what was expected. The best possible rating is one, zero is the worst result, all values in between are possible. Ratings for a worker w, be it negative or positive, increase the confidence $cnfd(w)$ and update the observed performance $pfmc(w)$. If the rating is below a threshold of 0.3 the worker suspends for ten rounds, similar to a deadline violation. Hence, requesters with negative experiences tend to make less usage of the crowdsourcing marketplace. In addition to the pure task description requesters announce the maximum price they are willing to pay for the processing of the task. Prices are also represented by random values within the range $[0,1]$. Tasks with high quality and high expected duration are more likely to have costs close to the maximum value.

Worker Behavior. When asked for a bid during an auction a worker first checks whether it is realistic to finish the task before the deadline considering all tasks the worker is working on. Each task has an expected duration t and each worker would only submit a bid if s/he has at least $1.5 \cdot t$ of time to work on the task before the end of the deadline considering already accepted tasks. The actual processing time is set to a random value within $[t, 2t]$. For workers with high real performance the processing time is more likely to be close to t. This value is set by the simulation environment and the workers do not know about the exact processing time in advance. For the evaluation we consider workers with two different bidding behaviors: conservative and aggressive. Conservative workers determine the price according to a linear combination of a tasks effort (i.e., a normalized value of the expected duration), the workers real performance,

and her/his workload. The rationale is that workers want more money for work-intensive tasks; workers with a high real performance are aware of their capabilities which influences the price as well. Finally, the higher a worker's current workload the more payment is conceived.

$$bid(w,t)_{conservative} = 0.4 \cdot effort(t) + 0.4 \cdot pfmc_{real}(w) + 0.2 \cdot load(w)$$

Aggressive workers, in contrast to conservative workers, do not increase the bid's price based on the workload but are more strongly driven by their own real performance.

$$bid(w,t)_{aggressive} = 0.4 \cdot effort(t) + 0.6 \cdot pfmc_{real}(w)$$

Whether a worker is conservative or aggressive is chosen randomly with the same probability. The processing of a task has a positive influence on the real performance of the worker w, i.e, $pfmc(w)_{real,new} = pfmc_{real}(w) + 0.1 \cdot (1 - pfmc_{real}(w))$. This modeling of a training effect results in a high learning rate for workers with low real performance and a slowed down learning effect for workers who are already very good.

Auction Processing. Auctions are conducted for the purpose of matching a task to a worker. As described in Section 4 there is a qualification and preselection stage before the actual auction in order to avoid spamming a huge worker base with auction request for which many workers may not have the necessary skills. Since we only consider one skill and have a limited number of 500 workers it is reasonable to admit all of them to the auctions. To achieve that the aptitude function, see Eq. 1, is set as follows:

$$aptitude : w \mapsto 1$$

After receiving the workers' bids they are ranked by a ranking function as defined in Eq. 2. Since there is only one skill the function is slightly adjusted:

$$rank : (w,b) \mapsto 0.6 \cdot pfmc(w) + 0.3 \cdot cnfd(w) + 0.1 \cdot (1 - price(b)).$$

Workers may either return a bid or refuse to submit a bid. From the received bids all values are removed whose price is higher than the price the requester is willing to pay. The remaining valid bids are ranked such that a high observed performance, high confidence, and a low price of the bid positively influence the rank. The emphasis at that stage clearly is on the performance and not on the price. It may happen that there is no valid bid; in that case the requester is informed that the task could not be processed.

Skill Evolution. In this work we want to investigate how crowdsourcing can benefit from skill evolution, which is achieved by assigning assessment tasks to workers. This is especially useful for workers with a low confidence value. For these workers only few or no ratings are available. An assessment task is a task that is assigned to a worker although another worker has won the auction and was assigned to the task as well. The workers are not aware of the fact that there are other workers processing the very same task; requesters are not either.

The crowdsourcing provider is responsible for paying for the training tasks. As usual, the result of the highest ranked worker is returned to the requester but it is additionally used as a reference for the training task. This enables the marketplace to generate a rating for the assessed worker by comparing her/his result to the reference. A further positive effect is the training of the assessed worker. The assignment of training tasks is based on the received list of valid bids. For controlling the skill evolution Equation 3 needs to be set accordingly.

The following definition of the assessment function, which results in disabling skill evolution and leads to purely profit driven auction decisions, maps each combination of workers, bids, and reference workers to 0.

$$assess_{profit} : (w, b, w_r) \mapsto 0.$$

In the evaluation we have used the following setting for the skill evolution enabled auctions.

$$assess_{skill} : (w, b, w_r) \mapsto \begin{cases} 0, & \text{if working queue not empty} \\ & \text{or } pfmc(w_r) < 0.8 \\ & \text{or } cnfd(w_r) < 0.8 \\ select(w, b), & \text{otherwise} \end{cases}$$

The function $assess_{skill}$ guarantees that only workers with empty working queue are assessed and that reference workers have high performance and high confidence. This is crucial because the worker w is rated according to the result of the reference worker w_r. If workers with a performance lower than 0.8 or confidence lower than 0.8 win an auction a training task assignment is prohibited. If all prerequisites are met the *select* function determines the workers who are assigned a training task. It is possible that multiple training tasks are assigned.

$$select : (w, b) \mapsto \begin{cases} 1, & \text{with probability } (1 - cnfd(w)) \cdot urg \\ 0, & \text{otherwise} \end{cases}$$

The *select* function assigns a training task based on the confidence of the considered worker. A low confidence increases the likelihood for a training task. The constant urg can be used to finetune the training task assignment procedure. In our experiments it is set to a value of 0.01. A high value raises the probability of assessment tasks.

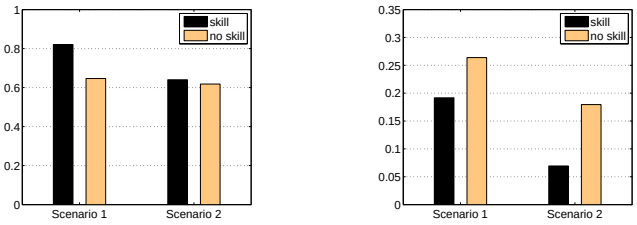
Discussions. Based on the introduced scenarios and simulation parameters, we test the benefit of skill evolution (*skill*) compared to regular auction processing (*no skill*). In the simulations, requesters issue a number of tasks to be processed by the crowd. However, requesters suspend their activity if the task quality is low (observed by low ratings) or task deadlines are violated. We hypothesize that a higher quality of task processing, and thus received ratings, also has positive effects on the profit of workers and the crowdsourcing platform. Table 1 gives an overview of the task statistics in each scenario. The number of *issued* tasks is influenced by the requesters' satisfaction. *No bid* means that a task could not be assigned to any matching worker.

The column *timeout* counts the number of tasks that were not delivered on time. Finally, the number of *training* tasks is depicted in the last column. All entries have the form *skill/no skill*.

Tasks	Issued	No bid	Timeout	Training
Scenario 1	527/315	2/5	57/73	757/-
Scenario 2	1687/1641	19/26	556/579	1310/-

Table 1. Tasks in Scenario 1 and Scenario 2.

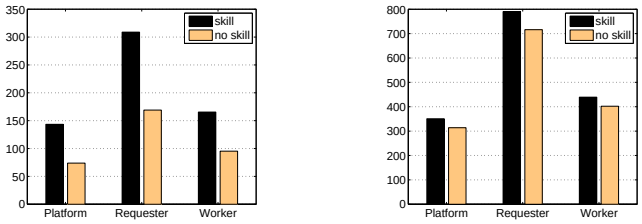
In Fig. 5, we compare the results of Scenario 1 and 2 on the basis of total rating scores given by requesters and the average difference between the evolved real performance of the workers and the observed performance.



(a) Rating. (b) Misjudgement.

Fig. 5. Rating and skill misjudgement.

For the *rating* (Fig. 5(a)) results in Scenario 1, the difference is more significant than in Scenario 2; 19% difference compared to 2%. In Scenario 1, whilst with no skill evolution support only an average rating score of slightly above 60% is given by the requesters, the advantage of skill evolution support is most apparent. In this scenario with low load, the system can optimally exploit the better accuracy of worker skill estimation, resulting in an average rating of 80%. Interestingly, the ratings are far lower in Scenario 2, although the average true performance of the workers is higher. The reason is that the high number of requesters causes a heavily loaded system and increases the probability of deadline violations. Also, low performing workers are more likely to win an auction. The reaction of requesters to deadline violation and low quality is to give bad ratings, in this case an average rating of 60%. Due to the heavy load the benefit of skill evolution is small because for determining an auction's winning bid, performance and confidence become less decisive but free working capacity is more important. For the *misjudgement* of the workers in Fig. 5(b) (lower values are better), the results indicate clear benefits of skill evolution. Misjudgement is based on the average difference between the real and the observed performance. With more tasks, and in particular assessment-tasks being issued, worker capabilities can be estimated more correctly. For Scenario 1, the difference is below 20% with and below 30% without skill evolution support. For Scenario 2 with more load, the results considering misjudgement are evidently better. With more transactions, the average performance values' difference in the skill evolution support model is around 7% and around 19% otherwise. Thus, assessments provide remarkable good results for reducing misjudgements.



(a) Scenario 1. (b) Scenario 2.

Fig. 6. Payments from requesters to workers and crowdsourcing platform.

In our simulation requester try to minimize expenses; workers and crowdsourcing marketplace try to maximize earnings. Currently, we use a simple model in which the marketplace collects for each transaction the difference between the maximum expenses, as specified by the requester, and the minimum salary, as specified by the worker bid. In a real setting, the crowdsourcing platform could decide to charge less for its service. The quality of the results, and thus, their satisfaction directly influences the task offering tendency of the requesters. Again, with skill evolution applied, more tasks are processed since good ratings encourage requesters in offering tasks at a constant rate (see Fig. 6(a)). Altogether, the requesters spend almost twice as much money with skill evolution. This is only true for Scenario 1 and similar to the previous results, the difference is far smaller considering overload situations. With more than six times as many requesters, their expenses remain way below the sixfold amount as spent in Scenario 1 with skill evolution support. In total, the expenses in Scenario 2 are almost the same with and without skill evolution. The ratios are similar for the benefits of the workers and the platform. As a summary, skill evolution generally performs better, however, is not antagonistic to overload scenarios. While with moderate task offering frequencies the model performs much better in all measurements, the differences become even when load increases and assessment-task further overload the platform. The results show that it is the responsibility of the platform to balance the task load and trade only with a fair amount of requesters.

6 Conclusions and Future Work

In this paper we present a novel crowdsourcing marketplace based on auctions. The design emphasizes automation and low overhead for users and members of the crowdsourcing system. We introduce two novel auctioning variants and show by experiments that it may be beneficial to employ assessment task in order to estimate members' capabilities and to train skills. Stimulating the demand for certain skills in such a way leads to skill evolution. As part of our ongoing research we plan to investigate how to allow for complex tasks and collaboration. Workers may, for instance, decompose tasks into subtasks, "crowdsource" the subtasks, and finally assemble the partial results into the final one. In such a setting the crowd would contribute the knowledge how to compose and assemble complex tasks. Furthermore, crowd members could provide higher level services such as

quality control and insurance. Apart from auction-based mechanisms, specifications such as WS-HumanTask [10] and BPEL4People [2] for modeling human interactions in service-oriented business environments need to be extended to cope with the dynamics inherent to open crowdsourcing platforms. For example, providing skill and quality models based on prior negotiated service-level agreements (SLAs) that augment WS-HumanTask's people assignment model.

Acknowledgement This work received funding from the EU FP7 program under the agreements 215483 (SCube) and 257483 (Indenica).

References

1. Agichtein, E., Castillo, C., Donato, D., Gionis, A., Mishne, G.: Finding high-quality content in social media. In: WSDM '08. pp. 183–194. ACM (2008)
2. Agrawal, A., et al.: WS-BPEL Extension for People (BPEL4People). (2007)
3. Amazon Mechnical Turk: http://www.mturk.com (last access March 2011)
4. Brabham, D.: Crowdsourcing as a model for problem solving: An introduction and cases. Convergence 14(1), 75 (2008)
5. Castellano, C., Fortunato, S., Loreto, V.: Statistical physics of social dynamics. Reviews of Modern Physics 81(2), 591–646 (May 2009)
6. CrowdFlower: http://crowdflower.com/ (last access March 2011)
7. DiPalantino, D., Vojnovic, M.: Crowdsourcing and all-pay auctions. In: EC '09. pp. 119–128. ACM (2009)
8. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Mass collaboration systems on the World Wide Web. Communications of the ACM To appear
9. eBay: http://tinyurl.com/5w6zgfg (last access March 2011)
10. Ford, M., et al.: Web Services Human Task (WS-HumanTask), Version 1.0. (2007)
11. Ipeirotis, P.G.: Analyzing the Amazon Mechanical Turk Marketplace. SSRN eLibrary 17(2), 16–21 (2010)
12. Klemperer, P.: Auctions: Theory and Practice. Princeton University Press (March 2004)
13. Kumar, R., Lifshits, Y., Tomkins, A.: Evolution of two-sided markets. In: WSDM '10. pp. 311–320. ACM (2010)
14. Leymann, F.: Workflow-based coordination and cooperation in a service world. In: CoopIS '06. pp. 2–16 (2006)
15. Schall, D., Dustdar, S.: Dynamic context-sensitive pagerank for expertise mining. In: SocInfo '10. pp. 160–175. Springer-Verlag (2010)
16. Schall, D., Truong, H.L., Dustdar, S.: Unifying human and software services in web-scale collaborations. IEEE Internet Computing 12(3), 62–68 (2008)
17. Su, Q., Pavlov, D., Chow, J.H., Baker, W.C.: Internet-scale collection of human-reviewed data. pp. 231–240. WWW '07, ACM (2007)
18. Vukovic, M.: Crowdsourcing for Enterprises. In: Proceedings of the 2009 Congress on Services. pp. 686–692. IEEE Computer Society (2009)
19. Yahoo: Answers scoring system. http://answers.yahoo.com/info/scoring_system (last access March 2011)
20. Yahoo! Answers: http://answers.yahoo.com/ (last access March 2011)

End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo

Anton Michlmayr, *Member, IEEE,* Florian Rosenberg, *Member, IEEE,*
Philipp Leitner, *Member, IEEE,* and Schahram Dustdar, *Member, IEEE*

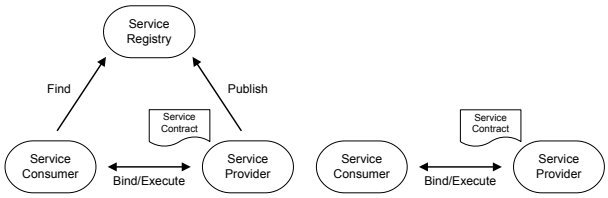
Abstract—Service-oriented Computing has recently received a lot of attention from both academia and industry. However, current service-oriented solutions are often not as dynamic and adaptable as intended because the publish-find-bind-execute cycle of the SOA triangle is not entirely realized. In this paper, we highlight some issues of current Web service technologies, with a special emphasis on service metadata, Quality of Service, service querying, dynamic binding and service mediation. Then, we present the Vienna Runtime Environment for Service-oriented Computing (VRESCo) that addresses these issues. We give a detailed description of the different aspects by focusing on service querying and service mediation. Finally, we present a performance evaluation of the different components, together with an end-to-end evaluation to show the applicability and usefulness of our system.

Index Terms—Web Services Publishing and Discovery, Metadata of Services Interfaces, Advanced Services Invocation Framework

1 Introduction

During the last few years, Service-oriented Architecture (SOA) and Service-oriented Computing (SOC) [1] has gained acceptance as a paradigm for addressing the complexity that distributed computing generally involves. In theory, the basic SOA model consists of three actors that communicate in a loosely coupled way as shown in Figure 1a. *Service providers* implement services and make them available in *service registries*. *Service consumers* (also called *service requesters*) query service information from the registry, bind to the corresponding service provider, and finally execute the service. Due to platform-independent service descriptions, one can implement flexible applications with respect to manageability and adaptivity. For instance, services can easily be exchanged at runtime and service consumers can switch to alternative services seamlessly, which increases organizational agility. Web services [2] represent the most common realization of SOA, building on the standards SOAP [3] for communication, WSDL [4] for service interface descriptions, and UDDI [5] for registries.

However, practice has shown that SOA solutions are often not as flexible and adaptable as claimed. We argue that there are some issues in current implementations of the SOA model. First and foremost, service registries such as UDDI and ebXML [6] did not succeed. We think this is partly due to their limited querying support that only provides keyword-based matching of registry content, and insufficient support for metadata and non-functional properties of services. This is also highlighted by the fact that Microsoft, SAP, and IBM have finally shut down their public UDDI registries in 2005. As a result, service registries are often missing in service-centric systems (i.e., no *publish* and *find* primitives). This leads to point-to-point solutions where service endpoints are exchanged at design-time (e.g., using E-mail) and service consumers statically bind to them (see Figure 1b).



(a) SOA Model (b) SOA Practice

Fig. 1: SOA Theory vs. Practice (adapted from [7])

Besides that, support for dynamic binding and invocation of services is often restricted to services having the same technical interface. In this regard, the lack of service metadata makes it difficult for service consumers to know if two services actually perform the same task. Furthermore, support for Quality of Service (QoS) is necessary to enable service selection based on non-functional QoS attributes such as response time (in addition to functional attributes).

In this paper, we discuss the issues we see in current SOC research and practice by describing the problems that arise when building SOC applications with current tools and frameworks. The main contribution is the presentation of the VRESCo service runtime environment that aims at solving some of these issues. To be more specific, the present paper focuses on service metadata, QoS and service querying, plus dynamic binding, invocation, and mediation of services. Additionally, we provide an extensive performance evaluation of the different components and an end-to-end evaluation of the overall

- *Anton Michlmayr, Philipp Leitner and Schahram Dustdar are with the Distributed Systems Group, Vienna Univ. of Technology, Argentinierstrasse 8, 1040 Vienna, Austria. E-mail: {lastname}@infosys.tuwien.ac.at*
- *Florian Rosenberg is with CSIRO ICT Centre, GPO Box 664, Canberra ACT 2601, Australia. E-mail: florian.rosenberg@csiro.au*

runtime, that shows the applicability of our approach.

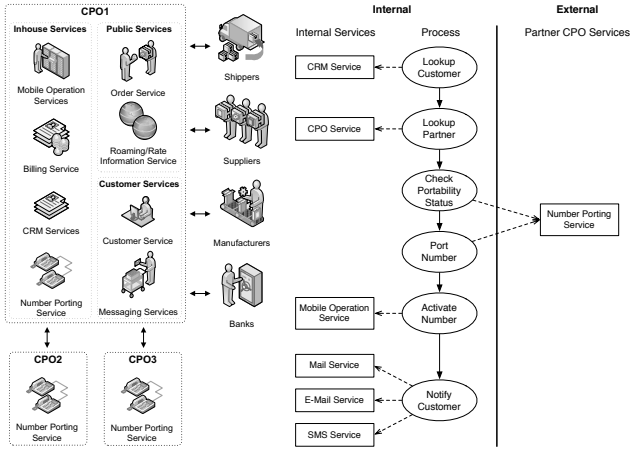
The remainder of this paper is organized as follows: Section 2 presents an illustrative example and summarizes some issues of SOC research and practice. Section 3 describes the details of the VRESCo runtime environment, while Section 4 gives a thorough evaluation of our work. Section 5 introduces related approaches and Section 6 finally concludes the paper.

2 Motivation and Problem Statement

This section first introduces a motivating example which is used throughout the paper. Then, we derive the problems developers face when engineering service-centric systems with current tools and frameworks.

2.1 Motivating Example

Figure 2 shows a typical enterprise application scenario from the telecommunications domain. The overview of this case study is depicted in Figure 2a.



(a) Case Study Overview (b) Number Porting Process

Fig. 2: CPO Case Study

Cell phone operator CPO1 provides different kinds of services: *Public Services* (e.g., Rate Information Service) can be used by everyone. *Customer Services* (e.g., SMS Service) are used by customers of CPO1, whereas *Inhouse Services* (e.g., CRM Services) represent internal services which should only be accessed by the different departments of CPO1. Besides that, CPO1 consumes services from its partners (e.g., cell phone manufacturers and suppliers) and competitors (e.g., CPO2 and CPO3). As discussed later, this scenario bears several challenges that are typical in service-centric software engineering.

According to European law, consumers can keep their mobile phone number when switching to another CPO. Figure 2b shows a simplified number porting process (depicted as oval boxes). This process is interesting because it contains both internal and external services (depicted as rectangles), and multiple service candidates. After the customer has been looked up using the CRM Service, the external Number Porting Service of the old CPO has to be invoked. If the number is portable the porting is executed by the old CPO. If this step was successful the new CPO is informed, which activates the new number using the Mobile Operation Service. Finally, a notification is sent to the customer using the preferred notification mechanism (e.g., SMS, E-mail, etc.).

2.2 SOC Challenges

Adaptive service-oriented systems bring along several distinct requirements, leading to a number of challenges that have to be addressed. In this section, we summarize the current challenges we see most important. The contribution of VRESCo is to address these challenges in a comprehensive service runtime environment.

- *Service Metadata.* Service interface description languages such as WSDL focus on the interface needed to invoke a service. However, from this interface it is often not clear what a service actually does, and if it performs the same task as another service. Service metadata [8] can give additional information about the purpose of a service and its interface (e.g., pre- and post-conditions). For instance, in the CPO case study without service metadata it is not clear if the number porting services of CPO2 and CPO3 actually perform the same task.
- *Service Querying.* Once services and associated metadata are defined, this information should be discovered and queried by service consumers. This is the focus of service registry standards such as UDDI [5] and ebXML [6]. In practice, the service registry is often missing since there are no public registries and service providers often do not want to maintain their own registry [7]. Besides service discovery, another issue is how to select a service from a pool of service candidates [9] by means of a querying language. For instance, CPO1 may want to select the SMS Service with the highest availability.
- *Quality of Service (QoS).* In enterprise scenarios QoS plays a crucial role [10]. This includes both network-level attributes (e.g., latency and availability), and application-level attributes (e.g., response time and throughput). The QoS model should be extensible to allow service providers to adapt it for their needs. Furthermore, QoS must be monitored accordingly so that users can be notified when the measured values violate Service Level Agreements (SLA).
- *Dynamic Binding and Invocation.* One of the main advantages of service-centric systems has always been the claim that service consumers can dynamically bind and invoke services from a pool of candidate services. However, in practice this requires identical service interfaces, which is often not the case. Therefore, we argue that the *bind* and *execute* primitives of SOA are not solved sufficiently. This raises the need for mechanisms that mediate between alternative services possibly having different interfaces.

Considering the CPO case study, the interfaces of CPO2's and CPO3's number porting service might differ, but the number porting process of CPO1 should still be able to seamlessly switch between them at runtime.

Besides these core challenges, other aspects such as *service versioning* [11] or *event processing* [12] are of crucial importance for SOC. However, a detailed description is out of scope of this paper, and the interested reader is referred to our previous work.

3 System Description

This section describes in detail the VRESCo runtime which was first sketched in [7]. Besides an architectural overview, we discuss service metadata and querying, as well as dynamic binding together with our service mediation approach.

3.1 Overview

The architectural overview of VRESCo is shown in Figure 3, which is adapted from [13]. The VRESCo core services are provided as Web services that can be accessed either directly using SOAP or by using the Client Library that provides a simple API. Furthermore, the DAIOS framework [14] has been integrated into the Client Library, and provides stubless, protocol-independent, and message-driven invocation of services. The Access Control Layer guarantees that only authorized clients can access the core services, which is handled using claim-based access control and certificates [13]. Services and associated metadata are stored in the Registry Database which is accessed using the Object-Relational Mapping (ORM) Layer. Finally, the QoS Monitor is responsible for regularly measuring the current QoS values. The overall system is implemented in C# using the Windows Communication Foundation [15]. Due to the platform-independent architecture, the Client Library can be provided for different platforms (e.g., C# and Java).

There are several core services. The Publishing/Metadata Service is used to publish services and metadata into the Registry Database. Furthermore, the Management Service is responsible for managing user information (e.g., name, password, etc.) whereas the Query Engine is used to query the Registry Database. The Notification Engine informs users when certain events of interest occur inside the runtime, while the Composition Engine [16] provides mechanisms to compose services by specifying hard and soft constraints on QoS attributes. In this paper, we focus on the main requirements for our client-side mediation approach which are the Metadata Service (including the models for metadata, services and QoS), the Query Engine, and the dynamic binding, invocation and mediation mechanisms.

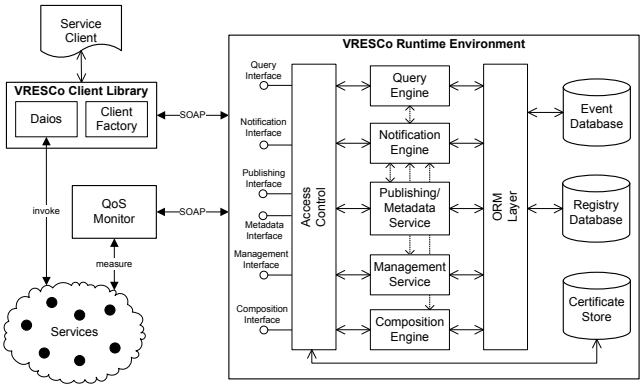


Fig. 3: VRESCo Overview Architecture

3.2 Service Metadata

The VRESCo runtime provides a service metadata model capable of storing information about services. This is needed to capture the purpose of services, which enables mediation between services that perform the same task. In this section, we describe service metadata and give examples from the CPO case study.

3.2.1 Metadata Model

The VRESCo metadata model introduced in [17] is depicted in Figure 4. The main building blocks of this model are *concepts*, which represent the definition of entities in the domain model. We distinguish between three different types of *concepts*:

- *Features* represent concrete actions in the domain that implement the same functionality (e.g., `Check_Status` and `Port_Number`). *Features* are associated with *categories* which express the purpose of services (e.g., `PhoneNumberPorting`).
- *Data concepts* represent concrete entities in the domain (e.g., `customer` or `phone_number`) which are defined using other *data concepts* and atomic elements such as strings or numbers.
- *Predicates* represent domain-specific statements that are either *true* or *false*. Each *predicate* can have a number of *arguments* (e.g., for *feature* `Port_Number` a *predicate* `Portability_Status_Ok(Number)` expresses the portability status of a given *argument* `Number`).

Furthermore, *features* can have *pre-* and *postconditions* expressing logical statements that have to hold before and after the execution of the *feature*. Both types of conditions are composed of multiple *predicates*, each having a number of optional *arguments*. These *arguments* refer to a *concept* in the domain model. There are two different types of *predicates*:

- *Flow predicates* describe the data flow required or produced by a *feature*. For instance, the *feature* `Check_Status` from our CPO case study could have the *flow predicate* `requires(Customer)` as *precondition* and `produces(Portability-Status)` as *postcondition*.

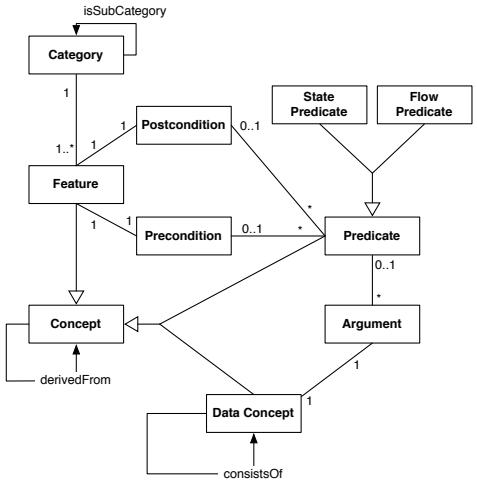


Fig. 4: Service Metadata Model [17]

- *State predicates* express global states that are valid before or after invoking a *feature*. For instance, *state predicate* `notified(Customer)` can be added as *postcondition* to *feature* `Notify_Customer`.

3.2.2 Service Model

The VRESCo service model constitutes the basic information of concrete services that are managed by VRESCo. The service model depicted on the lower half of Figure 5 basically follows the Web service notation as introduced by WSDL with extensions to enable service versioning and represent QoS on a service runtime level.

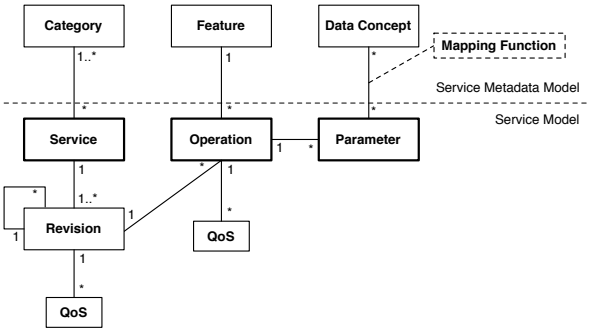


Fig. 5: Service Model to Metadata Model Mapping

A concrete *service* (e.g., Number Porting Service of CPO1) defines the basic information of a service (e.g., name, description, owner, etc.) and consists of a least one service revision. A *service revision* (e.g., the most recent version, or a stable one) contains all technical information that is necessary to invoke the service (e.g., a reference to the WSDL file) and represents a collection of *operations* (e.g, `Check_Status`). Every operation may have a number of *input parameters* (e.g., `Customer`), and may return one or more *output parameters* (e.g., `PortabilityStatus`). Revisions can have parent and child revisions that represent a complete versioning graph of a concrete service [11]. Both revisions and operations can have a number of *QoS attributes* (e.g., response time is 1200 ms) representing all service-level attributes as described below. The distinction in revision- and operation-specific QoS is necessary, because attributes such as response time depend on the execution duration of an operation, whereas availability is typically given for the revision (if a service is not available, all operations are generally also unavailable). In Section 3.5, we show how concrete services are mapped to the metadata and service model in order to perform service mediation.

3.2.3 QoS Model

Besides functional attributes described in the metadata model, non-functional attributes are also important. For instance, in our case study CPO1 may want to always bind to the Notification Service having the lowest response time. Therefore, QoS attributes can be associated with each service revision and operation in VRESCo. These QoS attributes can be either specified manually using the Management Service, or measured automatically (e.g., using the QoS Monitor introduced in [18]).

Attribute	Formula	Unit
Price	n/a	per invocation
Reliable Messaging	n/a	{true, false}
Security	n/a	{None, X.509,...}
Latency	$q_{la}(n) = \frac{1}{n}\sum_{i=0}^{n} q_{la_i}$	ms
Response Time	$q_{rt}(n) = \frac{1}{n}\sum_{i=0}^{n} q_{rt_i}$	ms
Availability	$q_{av}(t_0, t_1, t_d) = 1 - \frac{t_d}{t_1 - t_0}$	percent
Accuracy	$q_{ac}(r_f, r_t) = 1 - \frac{r_f}{r_t}$	percent
Throughput	$q_{tp}(t_0, t_1, r) = \frac{r}{t_1 - t_0}$	invocations/s

TABLE 1: QoS Attributes

Table 1 briefly summarizes the QoS attributes that are currently considered in VRESCo. Latency represents the time a request needs on the wire. It is calculated as the average value of n individual measuring points. Response time consists of the latency for request and response plus the execution time of the service. Availability represents the probability a service is up and running (t_0, t_1 are timestamps, t_d is the total time the service was down). Accuracy is the probability of a service to produce correct results where r_f denotes the number of failed requests and r_t denotes the total number of requests. Finally, throughput represents the maximum number of requests a service can process within a certain period of time (denoted as $t_1 - t_0$) where r is the total number of requests during that time. In addition to these pre-defined QoS attributes, users can define additional QoS properties for service revisions or operations.

3.3 Querying Approach

The VRESCo Query Language (VQL) provides a means to query all information stored in the registry (i.e., services and service metadata including QoS). In this section, we discuss the architecture of VQL followed by query specification and query processing.

3.3.1 Architecture

The VQL architecture was driven by the following requirements. First of all, declarative query languages such as SQL refer to database tables and columns, which makes queries invalid as soon as the database schema changes. Following the Query Object Pattern [19], queries can be built programmatically using query criteria that refer to classes and fields instead. These queries are finally translated into SQL statements, which makes them independent of the database schema. In this regard, VQL should provide such object-oriented querying interface and corresponding query expression library (similar to the Hibernate Criteria API [20]).

Moreover, it should be possible to define both mandatory and optional criteria by introducing different querying strategies that enable fuzzy or priority-based querying (e.g., services must have a response time below 500 ms and should be provided by company X). Finally, VQL queries should be type-safe (i.e., the query requester specifies the expected type of the query results) and secure (i.e., queries are protected against well-known security issues such as SQL injection).

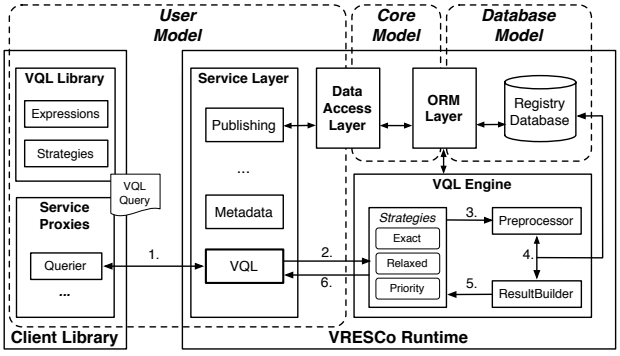


Fig. 6: VQL Architecture

The architecture of the VQL framework is shown in Figure 6. In general, the Client Library is used to invoke VRESCo core services (e.g., Publishing Service). Since these invocations represent remote method invocations, the Data Transfer Object pattern [19] is used to reduce the information sent from clients to the core services. Therefore, the VRESCo runtime operates on the *core model* (which represents the service metadata model introduced in Section 3.2), while clients operate on the *user model*. The task of the Data Access Layer (DAL) is to convert *core objects* to *user objects* and vice versa. The corresponding mapping between the two models is defined at design time using .NET attributes [21].

The advantage of this architecture is that clients operate on the *user model*, which represents a restricted view of the *core model*. Therefore, some information can be hidden from the clients (e.g., database IDs or versioning information for optimistic locking). Consequently, the VQL framework has to provide view-based querying, to be able to query on both models (depending on whether the query is issued client- or server-side). The task of the ORM Layer is then to map the entities of the *core model* to the *database model* (i.e., concrete database tables and columns), which is realized by NHibernate [20] using dedicated data access objects (DAOs).

According to this architecture, user queries are formulated using the Client Library, that provides an object-oriented querying interface to define query criteria, which is discussed in the next section. The query is then sent to the VRESCo runtime (step 1) and forwarded to the VQL Engine (step 2). The details of query processing (steps 3–5) are described in Section 3.3.3. Finally, the results are sent back to the query requester (step 6).

3.3.2 Query Specification

After describing requirements and architecture of the querying framework, we present how queries are specified. In general, VQL queries consist of six elements:

- *Return Type* R defines the expected data type of the query results. The return type needs to be an element of the VRESCo metadata model (e.g., a list of `Feature` objects).
- *Mandatory Criteria* C_m describe constraints which have to be fulfilled by the query (e.g., response time must be less than 500 ms).
- *Optional Criteria* C_o add constraints which should optimally be fulfilled but are not required (e.g., service provider should be company X).
- *Ordering* O can be used to specify the ordering of the query results (e.g., sort ascending by ID).
- *Querying Strategy* S finally defines how the query should be executed (e.g., exact or fuzzy matches).
- *Result Limit* L can be used to restrict the number of results (e.g., 10 or 0, which represents no limit).

The most important elements are criteria since they actually represent the constraints of the query. Moreover, criteria have different execution semantics depending on the querying strategy, which is discussed in Section 3.3.4. However, the main motivation is to allow the specification of mandatory and optional criteria.

In general, criteria consist of a set of *expressions* E that are used to define common constraints such as comparison (e.g., smaller, greater, equal, etc.) and logical operators (e.g., AND, OR, NOT, etc.). Table 2 shows criteria (C), expressions (E) and orderings (O) which are currently provided by VQL. Furthermore, the table indicates how each of these elements is translated to SQL, which is described in more detail later. It should be noted that VQL is extensible in that further expressions can be added easily.

Type	VQL	SQL	Description
C_m	Add	WHERE	Mandatory criteria
C_o	Match	IN/JOIN	Optional criteria
	And	AND	Conjunction of two expressions
	Or	OR	Disjunction of two expressions
	Not	NOT	Negation of an expression
	Eq	=	Equal operator
	Lt	<	Less operator
	Le	<=	Less or equal operator
E	Gt	>	Greater operator
	Ge	>=	Greater or equal operator
	Like	LIKE	Similarity operator for strings
	IsNull	IS NULL	Property is null
	IsNotNull	NOT NULL	Property is not null
	In	IN	Property is in a given collection
	Between	BETWEEN	Property is between two values
	Order	ORDER BY	Ordering of query results
O	Asc	ASC	Ascending ordering
	Desc	DESC	Descending ordering

TABLE 2: VQL/SQL Translation

Listing 1 shows an example query for finding services that implement the `Notify_Customer` feature in our CPO case study. As described above, queries are parameterized using the expected return type. In this case, the type `ServiceRevision` (line 2) expresses that the result of the query is a list of service revisions. In our example, two `Add` criteria (lines 5–7) are used to state that services have to be active and that each service has to implement the `Notify_Customer` feature (by using the `Eq` expression). The first parameter of expressions is usually a string representing a path in the user or core model (e.g., `Service.Owner.Company` describes the company property of the service owner). These strings are central to VQL, and are referred to as *property paths*. Additionally, three `Match` criteria are added in the example (lines 8–14). The first criterion expresses that services provided by *CompanyX* are preferred, while the second criterion defines that revisions should have tags starting with 'STABLE' (`Like` expression). The third criterion specifies an optional QoS constraint on response time, which should be less than 1000 ms. The operator '`&`' in line 13 represents a shortcut for an `And` expression. All three `Match` criteria use priority values as third parameter to define the importance of a criterion.

```
1 // create query object
2 var query = new VQuery(typeof(ServiceRevision));
3
4 // add query criteria
5 query.Add(Expression.Eq("IsActive", true));
6 query.Add(Expression.Eq("Service.Category.Features.Name",
7                         "NotifyCustomer"));
8 query.Match(Expression.Eq("Service.Owner.Company",
9                            "CompanyX"), 1);
10 query.Match(Expression.Like("Tags.Property.Name",
11     "STABLE", LikeMatchMode.Start), 3);
12 query.Match(
13     Expression.Eq("QoS.Property.Name", "ResponseTime") &
14     Expression.Lt("QoS.DoubleValue", 1000.0), 5);
15
16 // execute query
17 var querier = VRESCoClientFactory.CreateQuerier(
18     "username", "password");
19 var results = querier.FindByQuery(query, 10,
20     QueryMode.Priority) as IList<ServiceRevision>;
```

Listing 1: VQL Sample Query

The query is finally executed (lines 17–20) by instantiating a `querier` object using the Client Factory, and invoking the `FindByQuery` method using the desired querying strategy (e.g., `QueryMode.Priority`). Furthermore, the result limit of the query is set in order to return only 10 results.

3.3.3 Query Processing

Query processing is illustrated in Figure 6. When the query is sent to the VQL Engine, the specified querying strategy is executed, which is implemented using the strategy design pattern [22]. The query is forwarded to the *Preprocessor* component (step 3), which is responsible for analyzing the VQL query and generating the corresponding SQL query. Next, a NHibernate session is created to execute the generated SQL query on the database (step 4). After execution, the *ResultBuilder* component takes the results from the NHibernate session context. Since these results represent *core objects*, they may have to be converted back into the corresponding *user objects* (i.e., if the return type refers to the *user model*). This is done dynamically by invoking the constructor of the corresponding object using reflection. For both models, however, the *ResultBuilder* guarantees type-safety of the results, which are finally sent back to the client (step 5).

Algorithm 1 $processQuery(R, C, S, O)$

```
1: if ( isUserObject(R) ) then
2:     R ←MapUserToCoreObject(R)
3: end if
4: assocInfo ← R
5: for all ( crit ∈ C ) do
6:     for all ( expr ∈ GetExpressions(crit) ) do
7:         assocInfo←assocInfo ∪ ResolveAssoc(expr)
8:         propInfo← params ∪ ResolveProp(expr)
9:     end for
10: end for
11: query ← BuildFrom(assocInfo, propInfo, S)
12: query ← BuildWhere(query, assocInfo, propInfo, S)
13: query ← BuildOrder(query, O)
14: return query
```

Algorithm 1 depicts the pseudo-code of the *Preprocessor*. If the query refers to the *user model*, it is first transformed to the *core model* (lines 1–3). The *Preprocessor* then iterates over all criteria and expressions (lines 5–10). The *ResolveAssoc* function recursively analyzes the property paths of each expression to determine the necessary table joins. Similarly, the *ResolveProp* function extracts the property values of each expression. To give an example, reconsider line 8 of Listing 1: The property path `Service.Owner.Company` represents two associations `Service` and `Owner` that will be resolved using joins, and one property `Company` that will be compared with the expression's property value *CompanyX*. The concrete association/table and property/column names are retrieved using the ORM Layer. The collected information is finally used to build `FROM`, `WHERE` and `ORDER` clauses of the SQL query (lines 11–13), according to the VQL/SQL translation shown in Table 2.

3.3.4 Querying Strategies

The querying strategy influences how queries are executed. More precisely, it defines the *Preprocessor*'s behavior during SQL generation. The basic transformation process can be summarized as follows: `Add` criteria are transformed to predicates within the SQL `WHERE` clause, whereas `Match` criteria are handled as SQL sub-selects (`IN` or `JOIN`, see Table 2).

The *exact querying* strategy forces all criteria to be fulfilled, irrespective whether this is `Add` or `Match`. However, there are scenarios where `Match` has to be used instead of `Add` in order to get the desired results (i.e., by enforcing sub-selects using `IN` instead of `WHERE` predicates). In particular, when mapping N:1 and N:M associations (i.e., collection mappings in Hibernate terminology), a query cannot have the same collection more than once in the `WHERE` predicate. The use of sub-selects eliminates this effect in VQL, otherwise such queries would result in `null` since the associated tables are joined more than once. As an example reconsider the query in Listing 1 using the *exact* strategy. When having only one criterion with respect to `QoS`, `Add` can be used. However, if there would be a second QoS criterion, `Match` is required.

The *priority querying* strategy uses priority values for each criterion in order to accomplish a weighted matching of results. Therefore, each `Match` criterion allows to append a weight to specify its priority, which is internally added if the criterion is fulfilled. The query finally returns the results sorted by the sum of priority values. To give an example, the query in Listing 1 uses the priority values "1", "3" and "5". This means that the constraint on response time is more important than the constraint on revision tags. More precisely, queries that fulfill only the third `Match` criterion are preferred over queries that fulfill the first and the second `Match` criterion (since $5 > 3 + 1$).

The *relaxed querying* strategy represents a special variant of *priority querying* where each `Match` criterion has priority 1. Thus, this strategy simply distinguishes between optional and mandatory criteria. Results are then sorted based on the number of fulfilled `Match` criteria. This allows to define fuzzy queries by relaxing the criteria, which can be useful when no exact match can be found for a query. To achieve the necessary behavior, *relaxed* and *priority* querying both translate `Match` criteria into sub-selects using `JOIN` predicates.

3.4 Dynamic Binding

Dynamic binding is claimed to be one of the main advantages of SOA. In practice, however, services are often bound using pre-generated stubs that do not provide support for dynamic binding. Similar to querying strategies, we use the strategy pattern to implement a number of different rebinding strategies. We summarize all available strategies in Table 3.

Strategy	Proxy reconsiders binding...
`Fixed`	never
`Periodic`	periodically
`OnDemand`	on client requests
`OnInvocation`	prior to service invocations
`OnEvent`	on event notifications

TABLE 3: Rebinding Strategies

All rebinding strategies have their advantages and disadvantages. *Fixed* proxies are used in scenarios where rebinding is not needed (e.g., because of existing contractual obligations). *Periodic* rebinding causes background queries on a regular basis, which is inefficient if invocations happen infrequently. *OnDemand* rebinding results in low overhead but has the drawback that the binding is not always up-to-date. In contrast to this, *OnInvocation* rebinding guarantees accurate bindings but seriously degrades the service invocation time since service bindings are checked before every invocation. Finally, *OnEvent* rebinding uses the VRESCo Event Notification Engine [12] to combine the advantages of all strategies. Therefore, clients use subscriptions for defining in which situations to rebind, which is then triggered by events.

3.5 Service Mediation

Dynamic binding as described above naturally brings up the problem of how differences in service interfaces can be resolved at runtime. In this section, we introduce the VRESCo Mapping Framework (VMF) that handles the mapping from abstract features to concrete service operations (as described in Section 3.2), and perform mediation between different services that implement the same feature. The elements of the service model are mapped to our service metadata model as follows (see Figure 5): services are grouped into categories, where every service may belong to several categories at the same time. Services within the same category provide at least one feature of this category. Service operations are mapped to features, where every operation implements exactly one feature. However, we plan to provide support for more complex mappings using the VRESCo Composition Engine [16] (i.e., features will be represented as compositions of several service operations). The input and output parameters of service operations map to data concepts. Every parameter is represented by one or more concepts in the domain model. This means that all data that a service accepts as input or passes as output is well-defined using data concepts and annotated with the flow predicates `requires` (for input) and `produces` (for output). The concrete mapping of service parameters to concepts is described using mapping scripts, which will be discussed extensively below.

In general, the mediation approach follows the "feature-driven" metadata model. Therefore, a client that wants to invoke a service does not provide the input of the concrete service directly but in the conceptual high-level representation (i.e., the feature input in VRESCo terminology). The runtime takes care of lowering and

lifting the feature input and output, respectively. Lowering represents the transformation from high-level concepts into a low-level format (i.e., feature input to SOAP input) whereas lifting is the inverse operation (i.e., SOAP output to feature output).

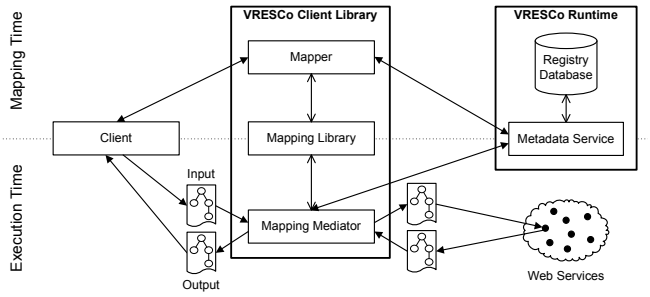


Fig. 7: VMF Architecture

Figure 7 shows an overview of the VMF architecture. Generally, VMF comprises two main components. Firstly, at mapping time, the *Mapper* component is used to create lifting and lowering scripts for each service. This information is stored in the VRESCo Registry Database using the Metadata Service. Secondly, at execution time, DAIOS is used as a dynamic service invocation framework. The *Mediator* component is used as an interceptor in DAIOS following the ideas presented in [23]. This mediator retrieves the lifting and lowering scripts from the VRESCo Metadata Service at runtime, and executes the corresponding mapping. This is done by applying all mapping functions sequentially, in the order they have been specified. In that sense, VMF implements an imperative, interpreted domain-specific language. In its current form, VMF does not optimize mapping scripts in any way.

Functions	Description
Assign	Link one parameter to another (source and destination must have the same data type)
Constants	Define simple data type constants
Conversion	Convert simple data types to other simple data types
Array	Create arrays and access array items
String	String manipulation operations (e.g., `substring`, `concat`)
Math	Basic mathematical and logical operations (e.g., `addition`, `round`, `and`, `or`)
CSScript	Define complex mappings directly in C#

TABLE 4: VMF Mapping Functions

Mapping scripts are defined using the *Mapping Library*, which includes a number of *Mapping Functions*. Mapping functions are the atomic building blocks from which all mapping scripts are constructed. We have summarized the provided mapping functions in Table 4 (grouped into 7 categories). Probably the most important function is `Assign`, which is used to map one input parameter or intermediary result to an output parameter (i.e., a Web service operation parameter in case of a lowering script, a feature output parameter in case of a lifting script). Functions from the *Constants* group are used to create new data directly in the mapping. All remaining mapping functions are used to transform parameters in various ways, e.g., from one data type to another, using string manipulation, or using mathematical and logical operations. Furthermore, more complex mappings can be defined in the CS-Script language [24]. Essentially, this allows to deploy custom mapping functions by using the full power of the C# programming language. For instance, this can be used to invoke external Web services at mediation time.

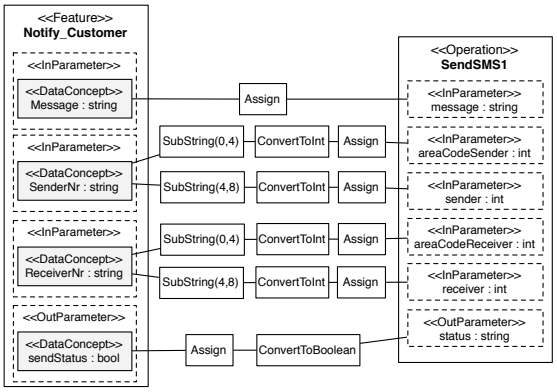


Fig. 8: VMF Mapping Example

We give a concrete mapping example in Figure 8. In this example, the abstract feature `Notify_Customer` from the CPO case study (see Section 2) is mapped to the concrete operation `SendSMS1`. The feature provides three input parameters and produces one output parameter. The parameter `Message` is identical in both interfaces, and can therefore be mapped directly (using only `Assign`). Note that for the `Assign` function to work both sides need to be represented using the same data concept (in this case string). The parameter `SenderNr` is split into the area code and the actual number. This is done using the string operation `SubString`, which takes the start index of the string and the length of the substring as parameters. Afterwards, both substrings are converted to integers using the `ConvertToInt` function. This is necessary since assigning a string to an integer is not possible. The `ReceiverNr` is handled similarly. So far, only input parameters have been mapped (i.e., all information given so far forms the lowering script for this service). The lifting script, which defines how the service output is mapped to the feature output, consists only of a `ConvertToBoolean` and another `Assign` function.

Listing 2 illustrates the first two mappings (`Message` and `SenderNr`) in C# code. Lines 4–5 show how the Mapper is created for feature `Notify_Customer` and operation `SendSMS1`. Both objects have to be queried beforehand (not shown in Listing 2 for brevity). The `Assign` function is again used as a connector to link the `Message` from the feature to the `Message` of the operation, whereas `mapper.AddMappingFunction()` adds the function to the mapping. Lines 14–21 get the area code from the feature's `SenderNr` as substring and convert it with the `ConvertToInt` function to an integer

```
1  // query NotifyCustomer and SendSMS1 instances using VQL
2
3  // create mapper from feature and operation
4  Mapper mapper = metadataService.CreateMapper(
5      NotifyCustomer, SendSMS1);
6
7  // map feature message to operation message
8  Assign messageAssign = new Assign(
9      mapper.FeatInParams[0],
10     mapper.OpInParams[0]);
11 mapper.AddMappingFunction(messageAssign);
12
13 // get AreaCode, convert to int and map it to operation
14 Substring acSenderStr = new Substring(
15     mapper.FeatInParams[1], 0, 4);
16 acSenderStr = mapper.AddMappingFunction(acSenderStr);
17 ConvertToInt acSenderInt = new ConvertToInt(
18     acSenderStr.Result);
19 acSenderInt = mapper.AddMappingFunction(acSenderInt);
20 mapper.AddMappingFunction(new Assign(acSenderInt.Result,
21     mapper.OpInParams[1]));
```

Listing 2: VMF Mapping Example Code

which is finally assigned to operation's input parameter `AreaCodeSender`. All further mappings from Figure 8 are implemented analogously.

4 Evaluation

In this section, we give an evaluation of the VRESCo runtime focusing on the topics covered in this paper. The purpose of this evaluation is twofold: Firstly, we show the runtime performance regarding service querying, rebinding, and mediation by using synthetic data. The main goal of this evaluation is to analyze the performance impact of each aspect in isolation. Secondly, we combine these aspects into a coherent end-to-end evaluation using an order processing workflow. The main goal is to understand the influence of each aspect with regard to the overall process duration in a realistic setting. Additionally, we show how the individual results of the first part interrelate in an end-to-end setting. All experiments have been executed on an Intel Xeon Dual CPU X5450 with 3.0 GHz and 32GB RAM running under Windows Server 2007 SP1. Moreover, we use .NET v3.5 and MySQL Server v5.1.

For mediation, rebinding and end-to-end evaluation we have created different sets of test services and QoS configurations (with varying response times) using the Web service generation tool Genesis [25]. These testbeds are described in detail in the corresponding subsections.

4.1 Querying Performance

First of all, we show the performance of the VQL Engine, which has been measured using the query shown in Listing 1. The test data are generated automatically: In every step, 5 categories are inserted, each having 5 alternative services with 10 revisions, while every revision has 1 tag and 11 QoS attributes with random values. It should be noted that in every step 20% of all services match the queried feature `Notify_Customer` and service owner *CompanyX*, while only 2% of all service revisions match all query criteria. To eliminate outliers, the results represent the median of 10 runs, while the database and Hibernate session cache are cleared after each run.

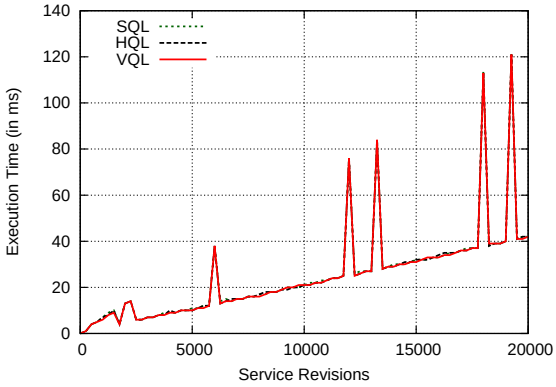


Fig. 9: Query Performance (NL)

Figure 9 compares the performance of the queries generated by SQL, HQL and VQL. Therefore, the query from Listing 1 was manually translated into HQL and SQL, while the VQL query is executed on *core objects* using the *exact* strategy without result limit (NL). The queries return only the ID of the matching revisions. Therefore, this table shows the performance of the native queries and does not include the time needed for converting the results back into `ServiceRevision` objects. The results indicate that the queries generated by all three approaches perform equally. In this regard, all approaches exhibit the same peaks, which are due to internal processing of the database.

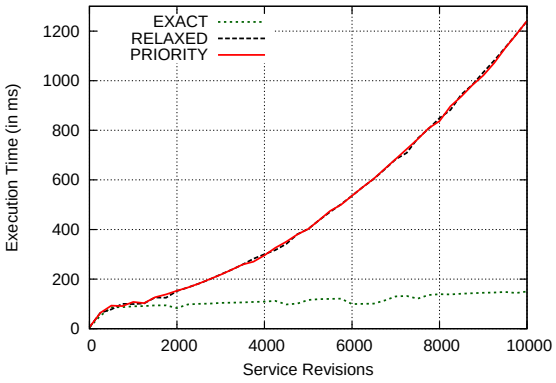


Fig. 10: Querying Strategies (User, L10)

Figure 10 compares the querying strategies using the same query on *user objects* and limited to 10 results (L10). The limit was chosen since *relaxed* and *priority* return more revisions than *exact* (which influences the results). It can be seen that *exact* is much faster than *relaxed*, while *relaxed* and *priority* have similar performance. The reason for the significant difference is that *relaxed* and *priority* use different table joins, and need to sum up and order by the total sum of priority values, while the query in *exact* mode can be optimized by the database.

Finally, Table 5 depicts the duration of the individual steps during VQL query processing. Therefore, the previous query is executed on both *core* and *user objects* using the *exact* strategy. Generation (G) indicates how long the *Preprocessor* needs to analyze and generate the query. Execution (E) depicts the actual query execution time, while Conversion (C) represents the time needed by the *ResultBuilder* to convert the query results.

Revisions	User			Core		
	G	E	C	G	E	C
1000	4,8	3,8	84,7	3,1	3,6	7,1
2000	4,8	14,6	87,5	3,2	14,4	6,8
3000	4,8	7,7	87,0	3,2	7,6	6,5
4000	4,8	9,8	77,5	3,2	9,7	6,5
5000	4,8	12,0	81,4	3,2	11,7	6,4
6000	4,8	13,5	83,7	3,1	13,5	7,0
7000	4,8	15,9	86,9	3,2	15,5	6,8
8000	4,8	17,9	86,3	3,2	17,6	7,3
9000	4,8	19,8	82,4	3,2	19,8	7,2
10000	4,8	22,2	86,6	3,1	20,5	6,8

TABLE 5: VQL Query Processing (in ms, User/Core, L10)

The results show that G is almost constant for *core/user objects*, while the latter is slightly slower since queries have to be translated to refer to *core objects*. Obviously, E is almost equal for both approaches. Finally, the table indicates that C is fast for *core objects*, while it takes some time for *user objects*. The main reason is that queries actually return IDs, while the corresponding entities are loaded from the NHibernate session context. Furthermore, revision objects have a number of collections (e.g., tags, QoS, etc.) that have to be converted by the *ResultBuilder* using reflection, which internally leads to a number of additional queries (since most collections are lazy-loaded [19]). In this setting, the time for C is constant for all revisions due to the result limit of 10.

4.2 Rebinding Performance

In the following subsection, we give an evaluation of the different rebinding strategies introduced in Section 3.4. For measuring the rebinding performance, we used GENESIS to simulate 10 services that implement the same feature. Then, we leveraged the QoS plug-in to continuously modify the response time of all services using a Gaussian distribution, and we additionally increased the variance after each step in order to simulate an environment where the QoS of services is subject to significant change. Finally, we implemented one client for each rebinding strategy and measured the average response time when invoking the service. As a result, we can see the impact of the different rebinding strategies for each client.

The results of this experiment are depicted in Figure 11. It should be noted that the response time of the best service is decreasing since we increase the variance. All services start with a (server-side) execution time of 2000 ms. The (client-side) response time differs about 400 ms which is caused by the network latency and the time needed for wrapping SOAP messages.

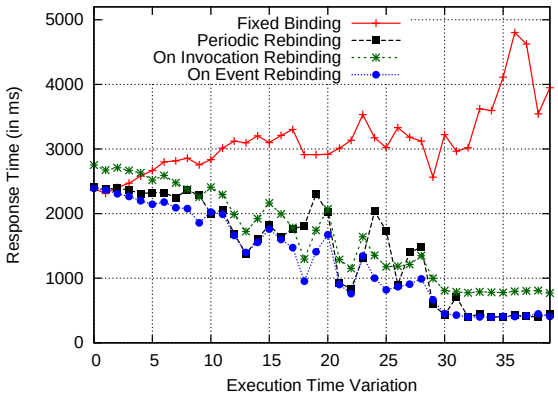


Fig. 11: Rebinding Strategies Performance

Obviously, clients with *fixed* binding usually perceive the worst response time because they are always bound to the same service. Clients using *periodic* rebinding mostly use services with good response time. However, since rebinding is done in pre-defined intervals the bindings are not always up-to-date (e.g., steps 17–18, 24–25, and 27–28 represent such situations). In contrast to that, clients with *OnInvocation* rebinding always invoke the best service since the rebinding is re-considered just before the service is invoked. However, this leads to a constant overhead of about 400 ms which is needed to check the binding and update if necessary. Finally, clients with *OnEvent* rebinding always bind to the best service without invocation overhead because the clients are notified asynchronously when the QoS changes and better services get available. However, the (optional) VRESCO eventing support must be turned on and the client needs a listener Web service. It should be noted that the performance of the Event Engine is sufficient which is detailed in [12]. Thus, all rebinding strategies have their strengths and weaknesses, and it depends on the specific situation which strategy to use.

4.3 Mediation Performance

Besides rebinding, we have also evaluated the overhead introduced by the VRESCO mediation facilities. We have again used the GENESIS tool for these tests.

Figure 12 depicts the response time of a single Web service invocation depending on the size of the message sent to the service. We have evaluated five different scenarios: (1) no mediation, (2) mediation using only constant mapping functions (replacing an input parameter with a constant string), (3) using mathematical functions (replacing a parameter with a calculated value), (4) using string modification functions (adding a constant string to a string parameter), and finally (5) using CS-Script (a simple script which exchanges the order of two parameters). Unsurprisingly, unmediated invocations are generally faster than any type of mediation. The performance of mediated invocations is similar no matter what type of mapping functions have been applied. However,

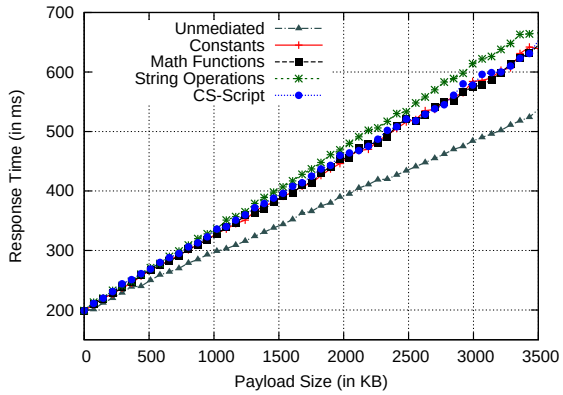


Fig. 12: Mediation Performance (Message Size)

in our experiments mediation using string operations introduces slightly more overhead than the other types. This is due to the fact that string operations naturally become more expensive when the strings become bigger.

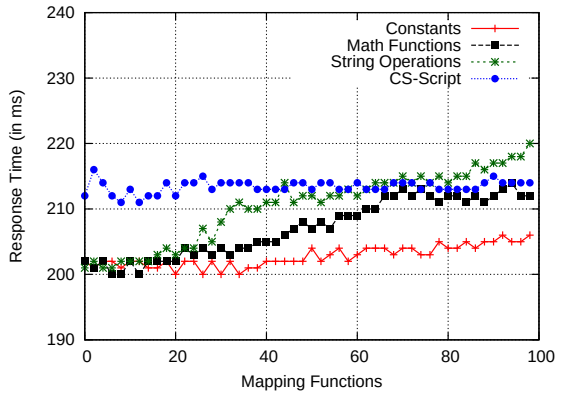


Fig. 13: Mediation Performance (Mediation Steps)

In Figure 13 we have studied the overhead introduced by different mapping functions in more detail. We have evaluated how the overhead introduced by mediation depends on the amount of mediation necessary (measured in the number of mapping functions applied). We have evaluated the same scenarios as before, but omitted the tests using unmediated invocations. Generally, the additional overhead introduced by a larger number of mapping functions is rather small: the difference between 1 and 100 mapping functions varies between 5 and 20 ms, which seems acceptable. As before, the overhead introduced by string operations heavily depends on the size of the strings to modify. Our experimentation string was rather sizable at 73 kByte, which explains the comparatively big overhead incurred by this type of mapping function. Note that the overhead of CS-Script mappings is constantly around 10 ms since the main overhead is the initialization of the scripting engine, while the execution of the actual script is negligible (as long as the script does not do any heavy computation, which would not be typical for mapping scenarios).

4.4 End-to-End Evaluation and Discussion

The end-to-end scenario combines all aforementioned aspects (i.e., querying, rebinding, mediation and invocation) into a larger order processing case study with the goal of ordering new cell phone contracts online (including mobile phone and SIM card). We implemented this workflow in C#. It consists of 19 overall activities split into 4 subprocesses. Basically, the process starts upon receiving an order via the company Web site. Afterwards the internal stock is checked for the availability of the phone and the SIM card. If one of those components is missing, it is ordered by using one of the internal or external suppliers, which is followed by a contracting subprocess. This subprocess creates a new contract and, if necessary, it adds a new customer to the CRM system. If the customer wants to transfer her old number, the number porting subprocess as depicted in Figure 2b is executed. Finally, the payment and shipping subprocesses are enacted and the cell phone number is activated in the GSM network.

The services used in the case study have been deployed on a different machine using GENESIS [25]. For each internal service (e.g., CRM, contracting) we have deployed only one alternative, whereas for each external service (e.g., Credit Card Service) multiple alternatives are available (between 60 and 250). For the internal notification service which is used to notify customers of their order status (using SMS, E-mail, mail, etc.) 30 alternatives are provided. This service is the only one that requires significant mediation. We use GENESIS to simulate a response time of 30–100 ms for each service.

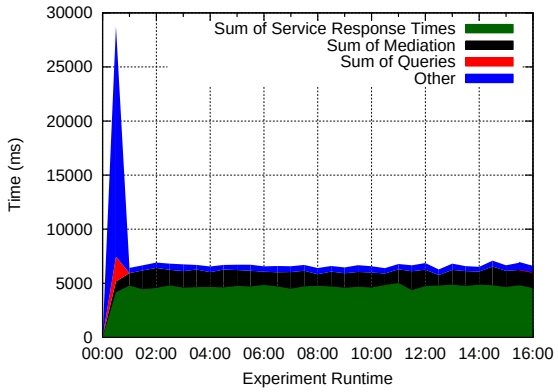


Fig. 14: End-to-End Performance

In Figure 14, we show the average process duration in this case study based on 40 concurrent clients running on one host that is also hosting the VRESCO environment. Each client continuously executes the process over an experiment time of 16 min. We have chosen the *Periodic* rebinding strategy for this scenario, to accommodate for our highly dynamic scenario with many alternatives for each external service. In order to get a big number of rebindings during our experiment time we have chosen a rebinding interval of 5 sec. The x-axis of the figure shows the experiment time (in minutes) and the y-axis

depicts the averaged process durations of the currently executing process instances. Right after bootstrapping the system, there is a steep incline in the overall duration because each client performs some initialization. This includes querying the available services (red part), as well as creating proxies and binding to one service candidate (blue part). Additionally, the services are invoked (green part) and a certain amount of mediation occurs (black part). After the initialization phase, the system stabilizes and the response times and mediation time are constant. The mediation overhead reflects our detailed mediation results from Figure 12. Together, service response times and mediation accounts for about 92% of the average process duration after the initialization phase. The remaining 8% (blue part) represent other factors such as thread handling or the workflow business logics. Please note that querying and an occasional rebinding still happens after the initialization phase, but it is no longer part of the average process execution times (on the y-axis). This is because the rebinding clients perform querying and rebinding asynchronously in a separate thread. Therefore, it solely depends on the rebinding strategy whether querying and rebinding is part of the process execution time or just part of the initialization phase (as shown in Figure 14). In case of the *OnInvocation* rebinding strategy, there would be querying and rebinding overhead in the overall process execution time, whereas for *OnDemand* and *OnEvent* the behavior would be similar as shown above.

Generally, the decision which rebinding strategy to use depends on the particular domain and the requirements. For example, for the Number Porting Service *fixed* binding is not a reasonable choice because even simple changes of the partner CPO's services (e.g., a different endpoint) would break the process. *OnDemand* is only reasonable if changes happen infrequently, and adaptation to changes is not time-critical. *Periodic* rebinding, on the other hand, is only adequate when services change frequently enough to warrant permanent polling for updates. Since number porting is not time-critical, we could have also used the *OnInvocation* rebinding strategy, which has a constant invocation overhead but always finds the best available service, or even better *OnEvent* which also eliminates this invocation overhead.

5 Related Work

In this section, we review related work concerning service repositories and service metadata, as well as service selection, invocation, and mediation.

Currently, several approaches and standards for service registries exist. We have compared some existing solutions with the VRESCo runtime, considering a carefully selected range of established standards, mature open-source frameworks and commercial tools. We consider the standards UDDI [5] and ebXML [6] (with special emphasis on the registry), Mule ESB and Galaxy repository [26], WSO2 ESB and registry [27], and IBM WebSphere [28] (including ESB, service registry and repository). Our comparison in Table 6 is structured according to the challenges introduced in Section 2.

Generally, all systems allow to store service metadata. Mostly, this is done in an unstructured way (e.g., using tModels in UDDI). There is only limited support for structured metadata in most approaches, whereas WebSphere provides an extensive structured metadata model (e.g., supporting OWL). To access data and metadata within the registry a query language or API is needed, which is provided by all approaches (WSO2 supports querying only based on Atom [29]). In contrast to VRESCo, type-safe queries are not supported by most approaches since querying is usually done on the unstructured service metadata model using languages such as SQL. Only WebSphere provides partial support by using XPath expressions for querying. Currently, explicit support for QoS attributes is not widely available – it is to some extent possible in WSO2 and WebSphere, and fully supported by VRESCo. WSO2 supports QoS only in terms of WS-Security and WS-ReliableMessaging. However, none of these frameworks except VRESCo provide QoS monitoring. Integration of dynamic binding, invocation and mediation of services is obviously not supported by pure registries such as UDDI or the ebXML registry. The other systems provide support in this respect due to their integrated ESBs. All systems except UDDI and VRESCo allow to store multiple versions of service metadata in the registry. However, only VRESCo provides end-to-end versioning support, which enables to seamlessly rebind and invoke different service revisions at runtime [11]. Finally, all approaches provide basic event notifications (e.g., if services are published) using E-mail, Web service notifications or Atom. Only WebSphere and VRESCo allow clients to subscribe to more complex events and event patterns using a rich subscription language.

Besides UDDI and ebXML, there are other standards for describing service metadata [8]. Some of them are used by semantic Web service approaches [30] (such as OWL-S [31], WSML [32] and SAWSDL [33]). It should be noted, however, that the VRESCo service metadata model introduced in Section 3.2 is not intended to compete with these approaches. We aim at enterprise development where metadata is an important business asset which should not be accessible for everyone, as opposed to the semantic Web service community where domain ontologies should be public to facilitate integration among different providers and consumers.

In general, several standards and research approaches have emerged that address the complexities of managing and deploying Web services [34]. In these approaches, service querying and selection play a crucial role, especially regarding service composition (e.g., [10], [35], [16]). However, the query models of current registries and Web service search engines [36] mainly focus on keyword-based matching of service properties which often do not cover the rich semantics of service metadata.

Challenge		UDDI	ebXML	Mule	WSO2	WebSphere	VRESCo
Service Metadata	Unstructured	+	+	+	+	+	∼
	Structured	∼	∼	∼	∼	+	+
Service Querying	Query Language/API	+	+	+	∼	+	+
	Type-safe Query	–	–	–	–	∼	+
Quality of Service	Explicit QoS Support	–	–	–	∼	∼	+
	QoS Monitoring	–	–	–	–	–	+
Dynamic Service Invocation	Binding & Invocation	–	–	+	–	∼	+
	Service Mediation	–	–	+	+	+	+
Service Versioning	Metadata Versioning	–	+	+	∼	∼	–
	End-to-End Support	–	–	–	–	–	+
Event Processing	Basic Notifications	+	+	+	∼	+	+
	Complex Event Processing	–	–	–	–	∼	+

TABLE 6: Related Enterprise Registry Approaches

Yu and Bouguettaya [37] introduce a Web service query algebra and optimization framework. This framework is based on a formal model using service and operation graphs that define a high-level abstraction of Web services, and also includes a QoS model. Service queries are specified as algebraic operators on functionality, quality and composition of services, and finally result in service execution plans. Optimization techniques are then applied to select the best service execution plan according to user-defined QoS properties. This work is complementary to ours: while the authors focus on their formal service model and introduce a query algebra for this model, we present a service runtime that provides end-to-end support for service management and querying functionality. Furthermore, we address dynamic binding and service mediation since service interfaces of different service providers are not always identical in practice. Dynamic binding of services has been addressed by other approaches (e.g., [38], [39]).

Pautasso and Alonso [38] discuss various binding models for services, together with different points in time when bindings are evaluated. They present a flexible binding model in the JOpera system where binding is done using reflection and does not require a specific language construct. Di Penta et al. [39] present the WS-Binder framework for enabling dynamic binding within WS-BPEL processes. Their approach uses proxies to separate abstract services from concrete service instances. Both approaches have in common that they rather focus on dynamic binding with respect to composition environments whereas VRESCo addresses binding at the core SOA level.

6 Conclusion

One of the main promises of SOC is the provisioning of loosely-coupled applications based on the publish-find-bind-execute cycle. In practice, however, these promises can often not be kept due to the lack of expressive service metadata and type-safe querying facilities, explicit support for QoS, as well as support for dynamic binding and mediation. In this paper, we have proposed the QoS-aware VRESCo runtime environment which has been designed with these requirements in mind. VRESCo offers an extensive structured metadata model and VQL as type-safe query language. Furthermore, we provide dynamic binding and mediation mechanisms that use pre-defined service mappings. We have evaluated our work regarding performance and discussed the results together with the experience gained in the CPO case study. The results show that the VRESCo runtime is applicable to large-scale adaptive service-centric systems.

As part of our ongoing and future work we want to link the VRESCo eventing [12] and composition [16] mechanisms. Furthermore, we envision to integrate SLA enforcement capabilities on top of VRESCo.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube). Additionally, we would like to thank Lukasz Juszczyk for providing the Web service testbed Genesis, and our master students Andreas Huber and Thomas Laner for their contribution to VRESCo.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, 2007.

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

[3] *SOAP Version 1.2*, World Wide Web Consortium (W3C), 2003, http://www.w3.org/TR/soap/.

[4] *Web Services Description Language (WSDL) 1.1*, World Wide Web Consortium (W3C), 2001, http://www.w3.org/TR/wsdl.

[5] *Universal Description, Discovery and Integration (UDDI)*, Organization for the Advancement of Structured Information Standards (OASIS), 2005, http://oasis-open.org/committees/uddi-spec/.

[6] *ebXML Registry Services and Protocols*, Organization for the Advancement of Structured Information Standards (OASIS), 2005, http://oasis-open.org/committees/regrep.

[7] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective," in *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07), co-located with ESEC/FSE'07*. ACM, 2007.

[8] D. Bodoff, M. Ben-Menachem, and P. C. Hung, "Web Metadata Standards: Observations and Prescriptions," *IEEE Software*, vol. 22, no. 1, pp. 78–85, 2005.

[9] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web*, vol. 1, no. 6, p. 6, 2007.

[10] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, May 2004.

[11] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "End-to-End Versioning Support for Web Services," in *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE Computer Society, 2008.

[12] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Advanced Event Processing and Notifications in Service Runtime Environments," in *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, 2008.

[13] ——, "Service Provenance in QoS-Aware Web Service Runtimes," in *Proceedings of the 7th International Conference on Web Services (ICWS'09)*. IEEE Computer Society, 2009.

[14] P. Leitner, F. Rosenberg, and S. Dustdar, "Daios – Efficient Dynamic Web Service Invocation," *IEEE Internet Computing*, vol. 13, no. 3, pp. 30–38, 2009.

[15] J. Löwy, *Programming WCF Services*. O'Reilly, 2007.

[16] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar, "An End-to-End Approach for QoS-Aware Service Composition," in *Proceedings of the 13th International Enterprise Computing Conference (EDOC'09)*. IEEE Computer Society, 2009.

[17] F. Rosenberg, P. Leitner, A. Michlmayr, and S. Dustdar, "Integrated Metadata Support for Web Service Runtimes," in *Proceedings of the Middleware for Web Services Workshop (MWS'08), co-located with EDOC'08*. IEEE Computer Society, 2008.

[18] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*. IEEE Computer Society, 2006.

[19] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[20] *Hibernate Reference Documentation v3.3.1*, Red Hat, Inc., 2008, http://www.hibernate.org/.

[21] J. Liberty and D. Xie, *Programming C# 3.0*. O'Reilly Media, Inc., 2007.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[23] P. Leitner, A. Michlmayr, and S. Dustdar, "Towards Flexible Interface Mediation for Dynamic Service Invocations," in *Proceedings of the 3rd Workshop on Emerging Web Services Technology (WEWST'08), co-located with ECOWS'08*, 2008.

[24] O. Shilo, "CS-Script – The C# Script Engine," 2009, http://www.csscript.net/.

[25] L. Juszczyk, H.-L. Truong, and S. Dustdar, "GENESIS - A Framework for Automatic Generation and Steering of Testbeds of Complex Web Services," in *Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'08)*. IEEE Computer Society, 2008.

[26] *Mule Galaxy, v1.5.1*, MuleSoft, Inc., Nov. 2009, http://www.mulesoft.org/display/GALAXY/Home.

[27] *WSO2 Registry, v2.0*, WSO2, Inc., Feb. 2009, http://wso2.org/projects/registry.

[28] *WebSphere Service Registry and Repository, v6.2*, IBM, Inc., Jul. 2008, http://www.ibm.com/software/integration/wsrr.

[29] R. Sayre, "Atom: The Standard in Syndication," *IEEE Internet Computing*, vol. 9, no. 4, pp. 71–78, 2005.

[30] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001.

[31] *OWL-S: Semantic Markup for Web Services*, World Wide Web Consortium (W3C), 2004, http://www.w3.org/Submission/OWL-S/.

[32] *Web Service Modeling Language (WSML)*, ESSI WSMO Working Group, 2008, http://www.wsmo.org/wsml/wsml-syntax.

[33] *Semantic Annotations for WSDL and XML Schema*, World Wide Web Consortium (W3C), 2007, http://www.w3.org/TR/sawsdl/.

[34] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and Managing Web Services: Issues, Solutions, and Directions," *The VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[35] J. Harney and P. Doshi, "Selective Querying for Adapting Web Service Compositions Using the Value of Changed Information," *IEEE Transactions on Services Computing*, vol. 1, no. 3, pp. 169–185, 2008.

[36] C. Platzer and S. Dustdar, "A Vector Space Search Engine for Web Services," in *Proceedings of the 3rd European IEEE Conference on Web Services (ECOWS'05)*. IEEE Computer Society, 2005.

[37] Q. Yu and A. Bouguettaya, "Framework for Web Service Query Algebra and Optimization," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 1, pp. 1–35, 2008.

[38] C. Pautasso and G. Alonso, "Flexible Binding for Reusable Composition of Web Services," in *Proceedings of the 4th International Workshop on Software Composition (SC'2005)*. Springer, 2005.

[39] M. D. Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. D. Nitto, "WS Binder: A Framework to Enable Dynamic Binding of Composite Web Services," in *Proceedings of the International Workshop on Service-oriented Software Engineering (SOSE'06)*. ACM, 2006.

Anton Michlmayr received the MSc degree in computer science from Vienna University of Technology in 2005. He is currently a PhD candidate and university assistant in the Distributed Systems Group at Vienna University of Technology. His research interests include software architectures for distributed systems with an emphasis on distributed event-based systems and service-oriented computing. More information can be found at http://www.infosys.tuwien.ac.at/Staff/michlmayr.

Florian Rosenberg is currently a research scientist at the CSIRO ICT Centre in Australia. He received his PhD in June 2009 with a thesis on "QoS-Aware Composition of Adaptive Service-Oriented Systems" while working as a research assistant at the Distributed Systems Group, Vienna University of Technology. His general research interests include service-oriented computing and software engineering. He is particularly interested in all aspects related to QoS-aware service composition and adaptation. More information can be found at http://www.florianrosenberg.com.

Philipp Leitner has a BSc and MSc in business informatics from Vienna University of Technology. He is currently a PhD candidate and university assistant at the Distributed Systems Group at the same university. Philipp's research is focused on middleware for distributed systems, especially for SOAP-based and RESTful Web services. More information can be found at http://www.infosys.tuwien.ac.at/Staff/leitner.

Schahram Dustdar is Full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Vienna University of Technology (TU Wien). He is also Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. Since 2009 he is an ACM Distinguished Scientist. More information can be found at http://www.infosys.tuwien.ac.at/Staff/sd.

Cost-Based Optimization of Service Compositions

Philipp Leitner *Member, IEEE,* and Waldemar Hummer *Member, IEEE,* and Schahram Dustdar *Senior Member, IEEE*

Abstract—For providers of composite services, preventing cases of SLA violations is crucial. Previous work has established runtime adaptation of compositions as a promising tool to achieve SLA conformance. However, in order to get a realistic and complete view of the decision process of service providers, the costs of adaptation need to be taken into account. In this paper, we formalize the problem of finding the optimal set of adaptations, which minimizes the total costs arising from SLA violations and the adaptations to prevent them. We present possible algorithms to solve this complex optimization problem, and detail an end-to-end system based on our earlier work on the PREvent (prediction and prevention based on event monitoring) framework, which clearly indicates the usefulness of our model. We discuss experimental results that show how the application of our approach leads to reduced costs for the service provider, and explain the circumstances in which different algorithms lead to more or less satisfactory results.

Index Terms—Service Composition, Service Level Agreements, Adaptation, Optimization

1 Introduction

Service-based applications have seen tremendous research activity in the last years, with many important results being generated around the world [1]. This global interest is justified by the ever increasing services industry, which is still only starting to explore the potential that new paradigms like Everything-as-a-Service (XaaS) or Cloud Computing provide [2]. However, to fully realize this potential, research and industry alike need to focus more strongly on non-functional properties and quality issue of services (generally referred to as QoS). In the business world, QoS promises are typically defined within legally binding Service Level Agreements (SLAs) between clients and service providers, represented, e.g., using WSLA [3]. SLAs contain Service Level Objectives (SLOs), i.e., concrete numerical QoS objectives, which the service needs to fulfill. If SLOs are violated, agreed upon monetary consequences go into effect. For this reason, providers generally have a strong interest in monitoring SLAs and preventing violations, either by using post mortem analysis and optimization [4], [5], or by runtime prediction of performance problems [6], [7]. We argue that the latter is more powerful, allowing to prevent violations before they have happened by timely application of runtime adaptation actions [8]–[10].

However, preventing SLA violations is, in general, not for free. For instance, some alternative services usable in a composition may provide faster response times (thereby improving the end-to-end runtime of the composite service, and reducing the probability of violating runtime related SLOs), but those services are often more expensive than slower ones. Therefore, there is an apparent tradeoff between preventing SLA violations and the inherent costs of doing so. We argue that this tradeoff is currently not covered sufficiently in research. Instead, researchers assume that the ultimate goal of service providers is to minimize SLA violations, completely ignoring the often significant costs of doing so (e.g., [9], [10]).

In this paper, we contribute to the state of the art by formalizing this tradeoff as an optimization problem, with the goal of minimizing the total costs (of violations and applied adaptations) for the service provider. We argue that this formulation better captures the real goals of service providers. Additionally, we present possible algorithms to solve this optimization problem efficiently enough to be applied at composition runtime. We evaluate these algorithms within our PREVENT (prediction and prevention based on event monitoring) framework [8].

The remainder of this paper is structured as follows. In Section 2, we motivate our work and present an illustrative example, which will guide us through the rest of the paper. Following in Section 3, we present our earlier work on prevention of SLA violations. In Section 4, we formalize the problem of cost-based optimization of service compositions. We explain possible algorithms to solve this problem efficiently in Section 5, which are experimentally evaluated in Section 6. Finally, we compare our work with the most important related scientific approaches in Section 7, and conclude the paper in Section 8.

2 Motivation

In this paper, we use the scenario depicted in Figure 1 (in BPMN [11] notation) to motivate and explain our approach.

• *Philipp Leitner, Waldemar Hummer and Schahram Dustdar are with the Distributed Systems Group, Vienna Univ. of Technology, Argentinierstrasse 8, 1040 Vienna, Austria. E-mail: {lastname}@infosys.tuwien.ac.at*

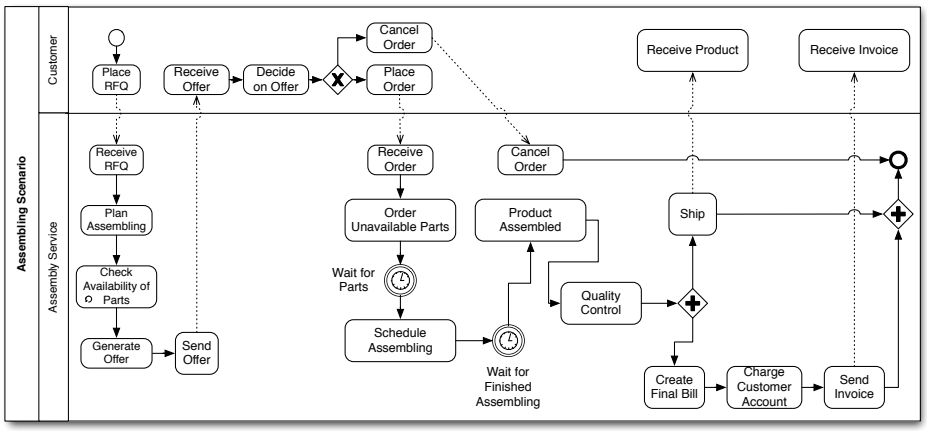


Fig. 1: Motivating Scenario

This scenario considers the case of a manufacturer of industry products. These products are constructed on-demand by assembling various parts, some of which can be produced in-house by the manufacturer, while others need to be ordered from external suppliers. The manufacturing process depicted in Figure 1 consists of two segments: firstly, the customer sends a request for quotation (RFQ), which the manufacturer responds to with an offer (consisting of estimated price and delivery time for the finished product), secondly, the customer can then order this product to the offered conditions. For reasons of brevity we concentrate on the two roles "Customer" and "Assembly Service" in the figure, even though the manufacturer interacts with many different external partners (e.g., suppliers of parts, shippers, credit card companies) to implement the described functionality. Since the manufacturer's business is based entirely on a service-based notion, the manufacturing process is implemented as a service composition, i.e., activities in the process are mapped to one or more invocations of (Web) services.

#	SLO Name	Description
1	**Time to Offer**	Time between receiving the RFQ and responding with an offer (in working days).
2	**Order Fulfillment Time**	Time between receiving the order and finishing the process (in working days).
3	**Process Lead Time**	Time between initializing the process and finishing it (excluding activities at customer side) in working days.
4	**Cost Compliance**	Cost overrun with regard to the offer in % of the offer.
5	**Product as Specified**	Product is exactly as specified.

TABLE 1: Service Level Objectives

With its key customers, the manufacturer has some established service level agreements. We provide a list of typical SLOs in Table 1. Note that these objectives can be of quantitative (SLOs #1 to #4) or of qualitative (SLO #5) nature.

All SLOs have some target values and penalties for violating these targets associated (see Table 2). Therefore, the manufacturer has a strong interest in complying to these SLOs, as long as the costs of doing so do not exceed the benefit. The manufacturer may apply a number of runtime adaptations to the process. We sketch some example adaptation actions in Table 3. The columns **+** and **-** refer to SLOs in Table 1, and indicate that the respective action has a positive (**+**) or negative (**-**) impact on this SLO. Note that these actions and impacts are just of examplatory nature, that is, while for some business cases outsourcing may reduce costs and increase the process duration (and error rate), this does not necessarily hold for all processes. Additionally, applying these actions generally also has some associated costs, which need to be taken into account (for instance, express shipping is more expensive than regular shipping). As we can see, for the manufacturer there is a tradeoff between the three dimensions duration, costs and quality, which is well-known in many fields of engineering.

#	Target Value	Costs of Violation
1	$<= 2$	Implicit costs - customer will choose a different manufacturer if offer is not received in time.
2	$<= 5$	Manufacturer grants 5% discount per 1 day delay, 20% max discount, not additive with SLO#3.
3	$<= 6$	Manufacturer grants 5% discount per 1 day delay, 20% max discount, not additive with SLO#2.
4	$<= 5$	Manufacturer cannot charge more than the offer plus 5%.
5	n/a	If wrong product is delivered, manufacturer needs to produce and ship the specified product within 7 working days and grant a 5% discount.

TABLE 2: Target Values and Penalties

Since the manufacturer business process is implemented as a service composition, applying these adaptations essentially boils down to adapting the service compositon. This can be done by either adapting the data flow of the composition (e.g., to use a different shipping option), by invoking different base services, or by changing the structure of the composition itself. In

#	Adaptation Action	+	-
1	Use faster shipper or faster shipping option, e.g., express shipping.	#2, #3	#4
2	Order more parts instead of producing them in-house.	#2, #3	#4
3	Generate offer with higher priority.	#1, #3	-
4	Outsource assembling and quality control.	#4	#2, #3, #5
5	Skip quality assurance or do it less thoroughly.	#2, #3, #4	#5
6	Add an additional quality assurance step.	#5	#2, #3, #4

TABLE 3: Possible Adaptation Actions

our previous work we have already shown how such adaptations can technically be applied [8], [9]. However, in these previous papers, the question of how the service provider can actually select these actions has not been discussed. Selecting the cost-optimal set of adaptations to prevent predicted violations results in an optimization problem, i.e., minimizing the total costs of all SLA violations plus all costs arising from the adaptation. This problem needs to be solved very efficiently, as the optimization has to be repeated at runtime for every composition instance that is predicted to violate one or more SLOs. Discussing this optimization problem is the main contribution of this paper.

3 Background

In order to provide some background information for this paper, we now present the PREVENT framework, which forms the basis for the research discussed here. Generally, PREVENT is a closed loop system [12] for self-optimizing service compositions. PREVENT is based on the existing SOA runtime environment VRESCo [13]. As we have sketched in Figure 2, the PREVENT framework consists of the seminal steps "monitor", "analyze", "plan" and "execute", as defined in the vision of autonomic computing [14]. We have previously presented our initial version of the PREVENT framework in [8].

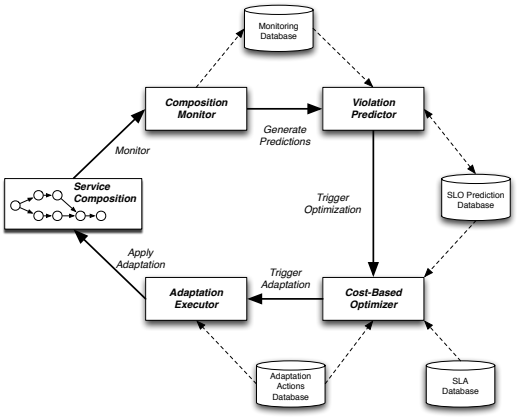


Fig. 2: Overall Framework

Generally, the idea of PREVENT is to use event-based monitoring of composition data to generate runtime predictions of SLA violations before they have happened. Based on these predicted violations, adaptation actions are triggered with the goal of preventing the violation. In this paper we focus on the implementation of the *Cost-Based Optimizer* component in Figure 2, which we have not discussed so far in our earlier work. For every composition instance, this component receives estimations of concrete SLO values from the *Violation Predictor* component, and decides (based on these estimations as well as on knowledge of standing SLAs and available adaptation actions), which adaptations should be applied to a composition instance. In the following, we refer to this decision procedure as *cost-based optimization*. We use the term *optimization time* as the point in time during a composition instance's execution at which cost-based optimization happens. The interested reader may download our current version of the PREVENT prototype[1].

3.1 Prediction of SLOs

Generally, the PREVENT approach to prediction of SLA violations is based on the idea of predicting concrete SLO values based on monitoring data. We distinguish three different types of information. *Facts* represent data which can already be measured at optimization time. *Unknowns* are the opposites of facts. They represent data which is entirely unknown at optimization time. Evidently, unknown data cannot be used in the prediction. *Estimates* are a kind of middle ground between facts and unknowns, in that they represent data which is not yet available, but can in some way be estimated. This is often the case for QoS data, since techniques such as QoS monitoring [15] can be used to get an idea of e.g., the response time of a service before it is actually invoked. The *Violation Predictor* uses both facts and estimates from previously monitored historical service executions to train a machine learning function (we use multi-layer artificial neural networks [16] for quantitative SLOs and C4.5 decision trees [17] for qualitative SLOs), which can then be used to produce a numerical estimation of the SLO values at runtime. More details about our approach to prediction of SLOs can be found in our earlier work [6].

We have sketched this machine learning based implementation of the *SLO Predictor* in Figure 3. One model is trained per SLO that needs to be predicted (even though the same model can be used if this SLO is used in multiple customer SLAs), and every model is trained from different data. Please see below for a discussion of how to identify which data to use for each SLO. Apparently, some historical executions of the service composition are necessary to bootstrap the training. The concrete amount of instances that are necessary depend both on the expected quality of prediction (more historical information in tendency improves the prediction quality) and on the size and complexity of the service

1. http://sourceforge.net/projects/vresco/

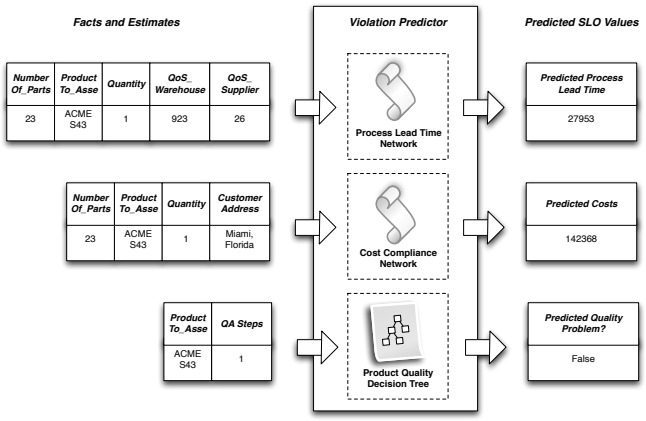


Fig. 3: Predicting SLOs Using Machine Learning

composition. This prediction approach cannot be used while no historical information is available. If this is the case, one could still use an alternative prediction approach, which is not data-based, e.g., [18]. However, a detailed discussion of this remains part of our future work.

For understanding the remainder of the paper, it is important to keep in mind that the *SLO Predictor* essentially implements a set of estimator functions, which can be used for any partially known instance of the composition (i.e., an instance, whose facts and estimates are partially known, for instance a half-finished instance) to generate an estimation of the SLO value when the instance is finished. We will use these estimator functions in our modelling in Section 4.

3.2 Identification of Factors of Influence

As input to the machine learning based *SLO Predictor* approach, we need to identify the most significant metrics that influence the SLO compliance of the composition. We refer to these metrics as the factors of influence of the service composition. Factors of influence are rarely obvious, even to domain experts. Hence, we have devised a process called dependency analysis, which can be used by business analysts to identify factors of influence. We summarize this process here, to the extent that is necessary for understanding the core contribution of the current paper.

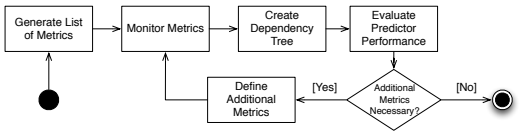


Fig. 4: Schematic Dependency Analysis Process

Dependency analysis is a semi-automated process. We rely on the domain knownledge of a human business analyst, but support her with automation and knowledge discovery tools to ease repetetive tasks. The high-level process is sketched in Figure 4. As a first step, the business analyst needs to define an (initial) list of potential factors of influence. These include both domain-specific metrics, which need to be defined manually, and typical QoS metrics, which can be automatically generated (e.g., for every used service we generate response time and availability metrics). For every potential factor of influence, a monitor is defined or generated, which specifies how this metric can be measured from a running instance. Secondly, a data set containing these factors needs to be generated, either by simulating the composition in a Web service test environment (e.g., Genesis2 [19]) or by monitoring real executions with monitoring of all potential factors of influence enabled. Using this data set, a dependency tree can be generated, as discussed in [5]. The dependency tree is essentially a decision tree, containing the factors that best explain SLO violations in the composition. The third step is then to use these factors to try and train a prediction model from the identified factors of influence. If this prediction model has a sufficiently high training data correlation against the measured data set (i.e., if the predictions generated with the predictor are highly correlated with the actual measured values), we can accept these factors and influence and use them in the *SLO Predictor* for the SLO. If the correlation is not sufficient, the business analyst needs to identify the reason for the lacking performance. Generally, the analyst will define additional potential factors of influence, and repeat from the second step.

3.3 Adaptation Actions

The PREVENT *Adaptation Executor* can execute a range of different adaptations of service composition instances. Generally, we distinguish three types of adaptations: data manipulation, service rebinding and structural adaptation. Data manipulation actions represent the most simple type of adaptation, where the composition is in fact not changed. Instead, the data flow of the composition instance is intercepted and some datum is changed (e.g., the *priority* parameter of the service invoked as part of the "ship" activity is changed to "high priority"). Service rebinding represents the common case where a different service is used to implement an activity in the composition, e.g., a faster shipping service is used in the activity "ship". For this type of adaptation, we differentiate between three types, one-to-one service rebinding without interface mediation (the original and the new service have identical interfaces), one-to-one service rebinding with interface mediation (the services have different interfaces, but the same number of service invocations is needed to achieve the required functionality), and substitution with subflow (the original service invocation is not only replaced with another single service invocation, but with a whole sub-composition). This adaptation is similar to the another type of adaptation, structural adaptation. In this case not only the data or service bindings of a composition change, but the logical structure of the composition itself.

This includes simpler cases like removing activities in an instance (e.g., skip the "quality control" activity) and more complex adaptations, where an entire subtree of the composition definition is replaced (e.g., outsource the assembling process to an external provider). Please refer to our earlier publications [8], [9] for details on how these actions are implemented. Most important for the remainder of this paper is to know how adaptation actions are defined in the PREVENT framework. We have sketched this in Figure 5.

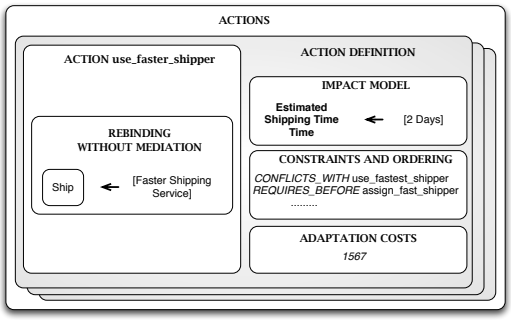


Fig. 5: Definition of Adaptation Actions

We can define any number of adaptation actions, which can be applied to an instance of the composition. Each of those definitions contains the description of the actual action, which can be any of the action types discussed above. In addition, the action definition also contains the impact model of the action, a list of constraints and ordering clauses, and the costs of applying this action. We assume that every adaptation action has a constant, non-negative cost. For example, the cost of using a faster shipping service is the cost of using the new service minus the costs of using the original shipping service. The impact model contains a set of impact clauses. Every impact clause represents the concrete impact that applying this adaptation action has on one concrete monitorable fact or estimate. Essentially, therefore, the clauses model updates to the data used to generate predictions (see Figure 3). Every adaptation action can have any number of positive as well as negative impacts on any fact or estimate. This impact value can be determined in several ways: (1) based on measured history data if the corresponding advice has already been used before, for example, using data mining; (2) based on SLAs with external providers, if such SLAs exist; or (3) by using QoS aggregation techniques [20]. We assume that the impact model specifies impact clauses for all metrics which the advice affects. Of course, impact clauses do not need to be exact (very often it will realistically be impossible to statically define an exact impact model before execution), however, more exact impact models lead to better predictions of SLOs after adaptation, which in turn leads to a better end-to-end performance of the PREVENT system.

4 Optimization Problem Formulation

In this section we formalize the problem of selecting the most cost-effective adaptation actions to prevent one or more predicted SLA violations. We consider an interaction of the service composition with a given client, who has a given SLA with the composition provider. Let I be the set of all possible composition instances of this client, and let $i \in I$ be concrete instances that we can monitor using the PREVENT tooling. Furthermore, let $S = \{s_1, s_2, \dots s_k\}$ be the set of SLOs defined in the relevant SLA. As part of the SLO definition, a penalty function is associated with all SLOs in S. Collectively, we refer to these functions as $P = \{p_{s1}, p_{s2}, \dots p_{sk}\}$. Penalty functions define the costs for the provider based on a measured SLO value, i.e., they are functions defined as $p_s : \mathbb{R} \rightarrow \mathbb{R}, s \in S$. Similarly, the measured value of an SLO m_s is a function $m_s : I \rightarrow [0 : 1]$. We normalize SLO values to the interval $[0 : 1]$ in order to make them comparable. Putting it all together, we define the penalty function for a given SLO s and instance i as $p_s^i \stackrel{\text{def}}{=} p_s(m_s(i))$. Penalty functions for SLOs can take many different shapes. The most important ones are (1) *constant* penalty (a constant payment needs to be made if a certain SLO threshold value is surpassed), (2) *staged* penalty (similar to a constant penalty, but with different levels of penalty), (3) *linear* penalty (the penalty is linearly increasing with the degree of violation), and (4) *linear* penalty with cap (the penalty is linearly increasing up to a maximum value). Even though these functions span many different types of mathematical functions, they share two essential characteristics. Firstly, SLA penalty functions are always monotonically increasing, i.e., $\forall p_s \in P : \forall x_1, x_2 \in \mathbb{R} : (x_1 < x_2) \implies (p_s(x_1) \leq p_s(x_2))$. This is evident, since the penalty for a higher degree of violation should never be smaller than the penalty for a lesser violation. Secondly, SLO penalty functions always have a point discontinuity in a special violation threshold point (t_1). Before (and including) t_1 the penalty is generally 0 (no violation has occured), and beyond this point a positive penalty needs to be paid ($\forall s \in S : \forall x \in \mathbb{R} : (x \leq t_1 \iff p_s(x) = 0) \wedge (x > t_1 \iff p_s(x) > 0)$). This also means that penalty functions are generally discontinuous. Furthermore, this property signifies that there is no incentive for the service provider to apply further adaptation and improve an SLO value below t_1, since all further improvements do not further reduce his costs (they are already 0 for this SLO).

To prevent violations, we are able to apply a number of possible adaptations to an instance i. We define $A = \{a_1, a_2, \dots a_l\}$ as the set of all possible adaptation actions, and $A^* \in \mathcal{P}(A)$ ($\mathcal{P}(A)$ denotes the powerset of A) as the subset of adaptation actions that are selected to be applied. We assume that all adaptations have some costs associated, defined as a cost function $c : A \rightarrow \mathbb{R}$. We assume that cost functions are constant, that is, we do not consider cross-pricing models for services [21], which would lead to non-constant costs of adaptation. Further-

more, adaptation actions, if applied, have some defined impact on the composition instance i. Hence, we define the transformation of i to a modified instance i' using the $\circ$ operator, defined as a function $\circ : I \times \mathcal{P}(A) \rightarrow I$. This is captured by the impact model, which has to be specified as part of the action definition (see Section 3).

Selecting the most cost-effective adaptation actions means finding the adaptation actions (A^*) that minimize the total costs for the service provider. The total costs TC are defined in Equation 1 as the sum of the costs of SLA violations after adaptation (VC) and the costs of adaptation (AC).

$$TC : \mathcal{P}(A) \rightarrow \mathbb{R}, TC(A^*) = VC(i \circ A^*) + AC(A^*) \quad (1)$$

AC is the sum of the costs of all applied adaptation actions (Equation 2).

$$AC : \mathcal{P}(A) \rightarrow \mathbb{R}, AC(A^*) = \sum_{a_x \in A^*} c(a_x) \quad (2)$$

VC is defined as the sum of all penalty functions applied to an instance (Equation 3).

$$VC : I \rightarrow \mathbb{R}, VC(i) = \sum_{s_x \in S} p^i_{sx} \quad (3)$$

Obviously, the goal of the service provider is to minimize TC. Hence, the optimization objective becomes finding the A^* that minimizes TC for a given instance i (Equation 4).

$$TC(A^*) = \sum_{s_x \in S} p^i_{sx} + \sum_{a_x \in A^*} c(a_x) \rightarrow min! \quad (4)$$

Note that we can easily calculate AC for any given A^*, but at optimization time VC is unknown (we do not know for sure which SLOs will be violated, with or without adaptation). However, the SLO Predictor provides estimations for SLOs based on instance data (see Section 3). Hence, we assume that we have estimation functions $e_s : I \rightarrow \mathbb{R}, s \in S$ available for each SLO, which estimate the concrete penalty values in advance with a reasonably small prediction error ϵ ($\forall s \in S, i \in I : |e_s(i) - p_s(i)| < \epsilon$). Replacing VC with its prediction using e_s leads to Equation 5, which we can solve.

$$TC(A^*) \approx \sum_{s_x \in S} e^i_{sx} + \sum_{a_x \in A^*} c(a_x) \rightarrow min! \quad (5)$$

However, not all combinations of adaptation actions are legal. Some adaptation actions are mutually exclusive (e.g., `use Shipping Service DHL` and `use Shipping Service UPS`), while others depend on each other (see our earlier work [9] for details on dependencies between adaptation actions). For simplicity, we capture these additional constraints using a penalty term $v : \mathcal{P}(A) \rightarrow \mathbb{N}$. The definition of v is shown in Equation 6.

$$v(A^*) = \begin{cases} \infty & A^* \text{ contains constraint violation} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

By incorporating this penalty term we arrive at our final target function (Equation 7).

$$TC(A^*) \approx v(A^*) + \sum_{s_x \in S} e^i_{sx} + \sum_{a_x \in A^*} c(a_x) \rightarrow min! \quad (7)$$

We have all necessary information to evaluate Equation 7 at optimization time for any set of actions A^*. However, finding the A^* that minimizes $TC(A^*)$ is still far from trivial, since Equation 7 is discrete and cannot be optimized analytically. We present algorithms to find a (near-)optimal solution in Section 5.

5 Algorithms

We will now discuss different approaches for finding solutions to this problem. These algorithms are implemented in the *Cost-Based Optimizer* component. Optimization is always triggered by a predicted violation of at least one SLO, and receives as input a list of monitored facts and estimates of the current instance.

5.1 Branch-and-Bound

Branch-and-Bound is a very general deterministic algorithm for solving optimization problems. The high-level idea of this approach is to enumerate the solution space in a "smart" way, so that at least some sub-optimal solutions can be identified and discarded prematurely, i.e., before they have been fully constructed and evaluated. We use a binary encoding to represent solutions, i.e., every solution is represented as a binary vector, and an adaptation action with index j is applied iff the solution vector is 1 at index j. For example, the solution vector 00110100 encodes that the third, fourth and sixth adaptation action should be applied. Evidently, $2^{|A|}$ different solutions exist for each optimization problem, where $|A|$ is the number of possible adaptation actions (but not all combinations need to be legal). For solutions that are still being constructed we allow a third symbol, "$*$", representing an action which is still undecided (alive). We refer to solutions, which contain at least one alive action, as partial, and solutions, which do not contain any alive actions, as complete. Therefore, the vector $001101 * 0$ is a partial solution, where the last-but-one action is alive.

We describe our general branch-and-bound algorithm in Figure 6. The algorithm is easy to understand. What is most important is the implementation of Line 13, the rules for pruning the search tree (i.e., for prematurely discarding solutions). In our branch-and-bound approach, we prune a partial solution in two cases: (1) if the partial solution already contains at least one conflict, or (2) if the partial solution already prevents all SLA violations (the penalty function p_s is 0 for all

```
1  # name: bab
2  # input: partial solution p,
3  #        next alive action index i,
4  #        target function v
5  # output: optimal complete solution
6
7  bab(p,i):
8    # recursion break condition
9    if(p is complete solution)
10     return p
11
12   # check if this sub-tree can be pruned
13   if(p is pruneable)
14         forall alive_actions(p) as j
15           set p(j) = 0
16     return p
17
18   # investigate solution sub-tree with p(i)=0
19   set p(i) = 0
20   s1 = bab(p,i+1)
21
22   # investigate solution sub-tree with p(i)=1
23   set p(i) = 1
24   s2 = bab(p,i+1)
25
26   # return better solution from both subtrees
27   if(v(s1) <= v(s2))
28     return s1
29   else
30     return s2
```

Fig. 6: Branch-and-Bound Algorithm

$s \in S$) without applying any more actions. Case (1) is trivial, since the target function value for all solutions in such a sub-tree will always be ∞. Case (2) lends itself to more discussion. Remember the assumption that every action has non-negative costs, and that we described SLA penalty functions as non-negative functions. Therefore, we can assure that for any solution where all penalty functions are 0, the additional application of more actions can never improve the target function value. Hence these partial solutions cannot be improved by applying more actions, and the remaining solution sub-tree can be pruned.

In Listing 6, we simply iterated over all actions in the order they appeared in the solution vector (in every step, we always just investigate the next action, see Lines 18 and 22). In general, this approach is suboptimal. Even though the order in which we investigate actions has no impact on the quality of our solution (the algorithm is deterministic, i.e., we will always find the global optimum eventually), the order may have an impact on the number of solutions we are able to prune. This is illustrated in Figure 7. Assume the following simple scenario: there is only one SLO, and 3 possible adaptations. Only adaptation 3 is able to prevent the violation of the SLO. Actions 1 and 2 have costs but no relevant influence. There are no conflicts between actions. Hence, the optimal solution vector is 001. In Figure 7(a), we strictly followed the algorithm in Listing 6 and investigated the actions in the order they appear in the solution vector. Since the only "useful" action is investigated last, we extend the whole solution tree without any pruning (the worst case, equivalent to full enumeration). Now, in Figure 7(b), we investigate the actions in reverse order (from back to front). Now, the "useful" action is investigated first, and a large part of this solution tree can be pruned according to pruning case (2).

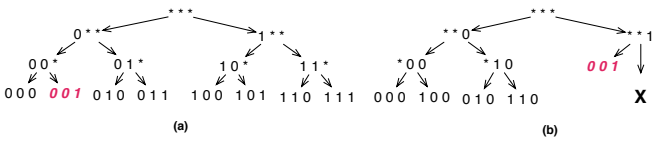


Fig. 7: Pruning of Solution Trees

Therefore, we can conclude that it is beneficial to investigate actions in a specific order that maximizes the number of solutions that can be pruned. We specify two possible criteria for this ordering: (1) the *impact* of an action on the SLOs (actions with higher total impact should be investigated first), and (2) the *utility* of an action (actions with higher utility should be investigated first). We will now define those two orderings.

Based on the set of historical process instances that we have already used to train the *Violation Predictors*, we can calculate an estimation of impact and utility of each action as follows. We define the set of available historical process instances as $H = \{h_1, h_2, \dots h_q\}$, with $H \subseteq I$. We refer to the number of historical instances as $q = |H|$. Now, we are able to calculate an estimation of the overall impact of an adaptation action a on a SLO s as $\Delta_{a,s}$ (Equation 8). Simply put, the impact is the arithmetic mean of the difference between SLO value with and without applying the adaptation to each historical instance.

$$\Delta_{a,s} = \sum_{h \in H} \frac{m_s(h) - m_s(h \circ \{a\})}{q} \tag{8}$$

Note that we have already defined in Section 4 that SLA penalty functions are monotonically increasing. Hence, higher impact values are generally good. However, the impact value may also be negative (i.e., $m_s(h) < m_s(h \circ \{a\})$). In this case this action has a negative impact on one of the SLOs, which is reasonble and realistic. For instance, an adaptation which reduces the process lead time can very well have a negative impact on the SLO cost compliance. Based on $\Delta_{a,s}$, we can now define the total impact of each action as the sum of its impact on all SLOs. Furthermore, we can define the utility of an action as its total impact devided by its costs. Now, we are able to improve the branch-and-bound algorithm trivially: instead of investigating the actions in the order they are specified in the solution vector, we now investigate them either in the order of their impact Δ_a (impact-based sorting), or in order of their utility u_a (utility-based sorting). We will evaluate and discuss both alternatives in Section 6, and compare them to Branch-and-Bound with randomly ordered actions.

5.2 Local Search

While the Branch-and-Bound algorithm discussed above has the advantage of always finding the optimal set of actions for any composition instance, the execution time of the algorithm increases exponentially with the number of available actions. Even though we can reduce the runtime using impact- or utility-based sorting of actions, the complexity still remains exponential. Hence, there is an evident need to find strong heuristics, i.e., non-deterministic algorithms that find "good" (even if not necessarily optimal) solutions in polynomial time.

A simple heuristic that is often used to very good ends is Local Search. Local Search is a metaheuristic, i.e., final solutions are constructed by iteratively improving a start solution. The general idea is that in each iteration the algorithm searches a specified *neighborhood* for better solutions than the current one. If at least one such solution is found, the algorithm progresses to the next iteration with one of the better solutions (typically the best one in the neighborhood, equivalent to steepest descent). If no better solution can be found in the neighborhood, the algorithm has converged to a local optimum and is terminated. Usually, this algorithm is repeated multiple times with different starting solutions (since different starting solutions can lead to different local optima). This kind of algorithm typically depends on the definition of (1) a suitable neighborhood and (2) a senseful selection of starting solutions. We use the following neighborhood definition: a complete solution vector is in the neighborhood of an original solution if the two solutions represented as binary vectors have a Hamming distance of 1, i.e., if they differ in exactly one bit.

```
1  # name: grasp_init
2  # input: number of start solutions n,
3           RCS max size r
4  # output: set of start solutions
5
6  grasp_init(n, r):
7    G = {} // empty set of start solutions
8    repeat n times:
9      pa = empty_partial_solution
10     while( VC(pa) > 0 ):
11       rcs = construct_rcs(pa, r)
12       if( empty(rcs) )
13         break
14       a = random(rcs)
15       pa(a) = 1
16     add(G, pa)
17   return G
```

Fig. 8: GRASP Construction Heuristic

For selecting the start solutions, we use two different approaches. The first and primitive one is to select n start solutions with m bits set to 1 at random. Alternatively, we propose to use an algorithm commonly referred to as GRASP [22] (greedy randomized adaptive search procedure). GRASP is essentially a variation of local search, in which the start solutions are constructed using a greedy heuristic. The idea is that GRASP can converge to a better solution than a simple local search because the start solutions are already better than random start solutions. However, some attention needs to be paid to using a greedy construction heuristic that actually generates start solutions, which are both of reasonable quality and at the same time widely spread over the search space.

We have sketched the construction heuristic that we have used in our implementation of GRASP in Figure 8. Summarizing, the algorithm constructs n solutions by stepwise addition of actions selected randomly from a restricted candidate set (RCS). The heuristic is based on similar concepts that we have already used in our discussion of Branch-and-Bound: the idea is to stop adding actions if either no more SLOs are violated or no senseful actions are available anymore (the RCS is empty), and to prefer adding actions which have a high utility value (u_a). Hence, in every step the RCS consists of the r (maximum size of the RCS) actions with highest non-negative u_a, which have not yet been added and which do not lead to a conflict.

5.3 Genetic Algorithm

As an alternative to locality-based heuristics (local search, GRASP) we also present a solution based on the concept of evolutionary computation. More precisely, we use genetic algorithms [23] (GA) as a more complex, but potentially also more powerful heuristic to generate good solutions to the cost-based optimization problem. The overall idea of GA is to mimic the processes of evolution in biology, specifically natural selection of the fittest individuals, crossover, and mutation. Therefore, in GA, we rather work on a population of solutions instead of a single one. We use the term "fit" to describe solutions with a good (low) target function value. Firstly, we generate a random start population. For this, we use the same primitive construction scheme as discusssed above for local search: we randomly apply m actions in every solution. Every following iteration of the algorithm (referred to as generations) essentially follows a three-step pattern.

Firstly, we *select* a set of solutions from the population to "survive" into the next generation. In our genetic algorithm implementation, the fittest solution (i.e., the one with the lowest target function value) is selected deterministically (elitism), while all remaining slots in the next generation population are selected using a process called tournament selection. In tournament selection, t random solutions from the last generation are put into a tournament. The fittest solution of the tournament is selected into the next generation. The parameter t steers the selection pressure: low t increases the time that the population takes to converge against a solution, but high t increases the danger of converging against a local optimum instead of the global one.

Secondly, *crossover* is used to produce new solutions based on the selected ones from the last generation.

The main challenge of implementing a strong crossover mechanism is to ensure that the crossover product of two fit solutions is also likely to be fit. Given the binary vector representation we use to encode solutions, we can make use of a simple one-point crossover scheme. We choose a random crossover point cp from $[1 : |A| - 1]$. To construct a new child, we copy the binary vector of the first solution from the start until cp, and the vector of the second solution from $cp + 1$ to the end of the vector.

This simple procedure ensures that characteristics of both original solutions are preserved. However, because of the random selection of cp, it is possible that the child solution has a conflict, even if this was not the case for any of the parents. In this case, we remove one of the conflicting actions at random.

Thirdly, we use *mutation* to introduce entirely new features into the population. The need for mutation can be illustrated easily: assume that a given action a is not applied in any solution in the population. Using one-point crossover as discussed above it is not possible to create any solution that uses a. Hence, we introduce some additional randomization. After crossover, we may randomly flip every bit in every solution in the population with a very small probability. This means that most solutions in the population are not mutated, but sometimes new actions are applied, which are not the product of crossover.

Name	Description	Default
Population Size	Number of solutions in every generation	150
Selection Pressure	Number of solutions to select for tournament selection	2
Crossover Probability	Probability per solution that crossover is applied	0.8
Mutation Probability	Probability per bit that mutation is applied	0.02
Break Condition	Condition for stopping the algorithm	No impr. in 20 iterations

TABLE 4: GA Configuration Parameters

GAs are notorious for having many parameters to fine-tune the performance of the optimization. For illustrative purposes, we have summarized the parametrization options available in our implementation of GA in Table 4, including some values that we found to provide useful default parameters if applied to the cost-based optimization problem. Evidently, further customization would also be possible, for instance by using a different selection or crossover scheme. Unless stated otherwise, we will use the configuration described in Table 4 for experimentation in Section 6.

Unfortunately, this "canonical" GA implementation takes a significant amount of time to converge against a solution, since the solution space is searched solely through the (rather unguided and strongly randomized) means of crossover and mutation. One possibility to improve this aspect is to combine the canonical GA with local optimization as presented above. This leads us to an adapted algorithm, which we have sketched in Figure 9. In literature, such combinations of GA and local search are often referred to as Memetic Algorithms [24] (MA).

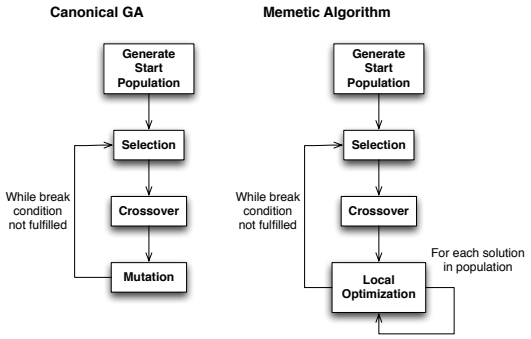


Fig. 9: Memetic Algorithm

The main changes of MA (as compared to GA) are as follows. Firstly, a new *Local Optimization* operator is introduced after crossover. Local optimization applies the local search algorithm as discussed above to each solution in the generation, basically reducing the population to a set of locally optimal solutions. Secondly, we remove the mutation operator from the algorithm (technically speaking, we set the mutation probability parameter to 0). The main reason is that given that all solutions in the population are already locally optimal, randomly mutating one bit in a solution can only lead to a worse solution. In theory it is possible that multiple bits in a single solution are mutated at the same time, and that these mutations lead to an improvement, but this corner case is very unlikely in practice. Furthermore, the main motivation for having mutation in the first place was that it is the only way of introducing new actions in the canonical GA. This is no longer the case, since local search can do the same thing.

Generally, MA is slower than GA, since more solutions are evaluated in each generation (evidently, MA executes one local search for every solution in each generation). However, the algorithm potentially converges against a very good solution in a low number of generations. Hence, we argue that in practice MA improves on the canonical form most of the time for our problem. This will be substantiated further in Section 6.

6 Experimentation

In the following section, we will numerically validate the algorithms discussed in Section 5 based on an implementation of the scenario presented in Section 2. For reasons of brevity, we only summarize the experiment setup here. More details can be found in the accompanying experimentation web page[2]. In addition, we do not explicitly evaluate the prediction quality (i.e., the SLO Predictor component) here. The interested reader may find a numerical evaluation of the prediction in [6], as well as in [8].

2. http://www.infosys.tuwien.ac.at/prototype/VRESCo/experimentation.html

We have implemented the scenario from Section 2 using .NET Windows Communication Foundation[3] (WCF) technology and the VRESCo SOA runtime environment on a server running Windows Server Enterprise 2007, Service Pack 2. The machine is equipped with 2 2.99GHz Xeon X5450 processors and 32 GByte RAM. In order to train PREvent, we have initialized the system with a set of 9796 historical composition instances. These instances were created by executing the service composition repeatedly. In this historical data set, 3660 instances have not been adapted, while one or more adaptation actions have been applied in the remaining 6136 instances. In our experiments, we consider the case of an SLA containing up to five SLOs, similar to the previous example. Note that we have used an integer value in $[0 : 15]$ to represent product quality in this example, in order to allow for more fine-grained distinctions of different levels of product faults. In Table 5 we have sketched these SLOs and their basic statistics. μ is the mean value of the SLO without adaptation. $\mu*$ is the mean among instances to which some adaptation has been applied. σ and $\sigma*$ are the respective standard deviations. As before, t_1 is the violation threshold. Furthermore, SLO 1 is associated with a staged penalty function with 9 stages, SLO 2 and 3 are both associated with fixed penalty functions, SLO 4 is associated with a linear penalty function with cap, and SLO 5 with a linear penalty function without cap. Additionally, we have defined 49 adaptation actions that have positive and negative influences on some or all of these SLOs. Every action has been associated with a positive cost value.

As a first experiment, we analyse the suitability of different variants of the Branch-and-Bound algorithm. As all of these algorithms are deterministic, we are guaranteed to find the optimal solution to any optimization problem eventually. However, the three different versions of the algorithm (Branch-and-Bound with random action sorting, with impact-based sorting, and with utility-based sorting) may differ significantly with regard to their runtime. As an independent measure of algorithm runtime, we use the number of solutions that have to be evaluated. All results concerning algorithms with randomized elements are arithmetic means of 5 repeated runs.

#	SLO Name	μ	μ^*	σ	$\sigma*$	t_1
1	Order Fulfillment Time	38811	35560	4708	6004	37000
2	Payment Time	4187	2202	28	1124	4150
3	Shipping Time	1285	864	144	347	1300
4	Product Quality	2.6	3	1.9	2.5	3
5	Cost Compliance	851	1149	212	521	1400

TABLE 5: Case Study SLOs

Figure 10 plots the number of solutions depending on the number of adaptation actions that are available (up to a maximum of 17 actions, note the logarithmic scale on the y-axis). For reasons of comparison we also plot local search in the figure, whose runtime grows linearly with the number of actions. It was not feasible to evaluate Branch-and-Bound for more than 17 actions.

3. http://msdn.microsoft.com/en-us/library/ms735967(VS.90).aspx

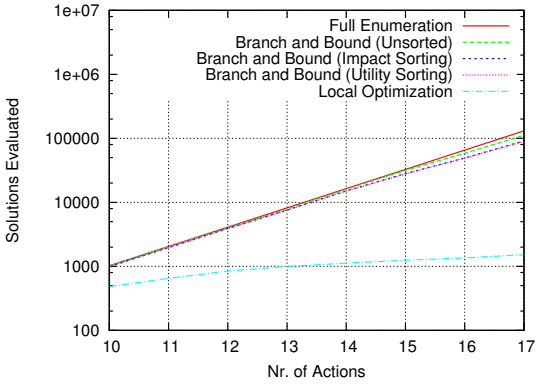


Fig. 10: Solutions Evaluated For Branch-and-Bound

As we can see, there is little difference between the three variants of Branch-and-Bound, and none is able to reduce the number of solutions that have to be evaluated significantly below full enumeration. The reason for this unsatisfying result is that, in this concrete optimization instance, very little combinations of actions can prevent the violation of all SLOs (the SLOs are conflicting), i.e., bounding condition 2 cannot be applied very often. We can see that, by relaxing the problem and disabling SLOs 4 and 5, a significant performance boost can be achieved (Figure 11) by both impact-based and utility-based sorting. The difference between impact-based and utility-based sorting is not significant.

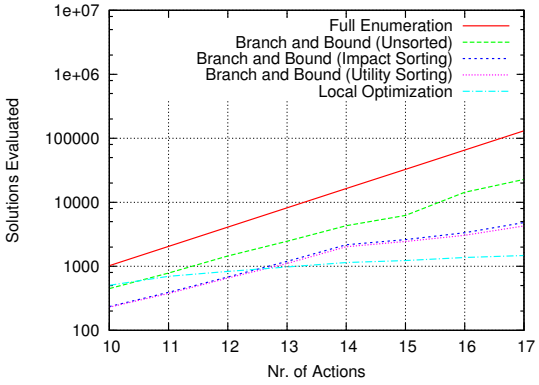


Fig. 11: Solutions Evaluated Without Conflicting SLOs

However, even though smart action sorting can reduce the solution space if there are no conflicting SLOs, the number of solutions that need to be evaluated still grows exponentially with the number of available actions. Hence, solving the cost-based optimization problem deterministically is only possible for very small problems. If the set of possible adaptations grows, we need to fall back to heuristic optimization. For these algorithms, there are no guarantees about the quality of the solution. That means that we need to compare them

in two dimensions. Firstly, and similar to before, we need to look at the number of solutions that are evaluated before the algorithm produces the final result (Figure 12), as a measure of the runtime of the algorithm. Secondly, we also need to take into account the quality of the best found solution (Figure 13).

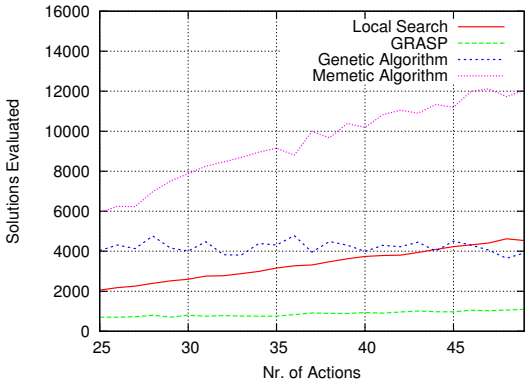


Fig. 12: Solutions Evaluated Per Heuristic Algorithm

In Figure 12 we can see that, not surprisingly, all algorithms scale much better than Branch-and-Bound (note the linear scale on the y-axis and compare with Figure 11). GRASP is very efficient, and the fastest algorithm in this experiment with an almost constant runtime. The computation of local search is also reasonably efficient, but the number of solutions that have to be evaluated increases more strongly as compared to GRASP. This is because for GRASP the start solutions are already better, hence less local search steps are necessary before a solution is reached. Note that the number of solutions evaluated for local search is directly proportional to the number of start solutions used. In this experiment we used 25 start solutions. If we had used 50 start solutions instead, the runtime of local search would have been almost on the level of MA. GA also has a relatively constant runtime, but on much higher level than GRASP. The slowest algorithm in this experiment is MA, which is due to its unique combination of local optimization and genetic algorithm.

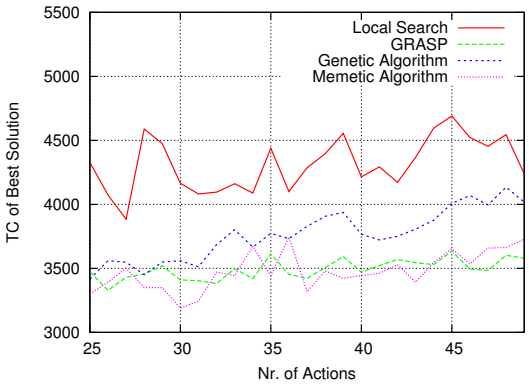


Fig. 13: Quality of Solution Per Heuristic Algorithm

In Figure 13, the algorithms are compared with regard to solution quality, measured as predicted total costs (TC, as defined in Section 4) for the service provider. We observe a quite clear ordering of algorithms in this experiment. GRASP and MA generally perform best. For most instances, MA is slightly better, even though this is not true for all cases. GA comes in third, and local optimization with random start solutions usually produces solutions vastly inferior to all competitors.

Drawing conclusions from these experiments, we note that Branch-and-Bound is applicable in situations where just a small set of actions is available. In general, impact-based or utility-based sorting should be used instead of random sorting, since there is no evident disadvantage to these approaches and they may be helpful if there are no conflicting SLOs. We did not discover a significant difference in the performance of these two variants. If more actions are available, MA and specifically GRASP are interesting candidate algorithms. GRASP produces good solutions in very little time and can generally be used even for short-running compositions where adaptation decisions need to be taken in a short time frame (below 1 second). MA is very promising in case of long-running compositions, where the time necessary to find a solution is not critical. MA often produces slightly better solutions than GRASP, but takes much more time to do so.

In a second set of experiments, we now evaluate the end-to-end effectiveness of PREVENT. That is, we analyze if the system fullfills its main promise, preventing SLA violations and reducing the total costs for the service provider. Hence, we execute 500 instances of the scenario composition and monitor the actual total costs and violations (after adaptation). We compare these numbers with the number of violations and the total costs that the PREVENT SLO Predictor can predict after roughly half of the service composition is finished. We assume that these predictions reflect the violations and costs that we would end up with if we did nothing at all. Since our case study is rather short-running, but uses a relatively large set of adaptations, we use GRASP for cost-based optimization. The results of this experiments are depicted in Table 6.

Evidently, the usage of PREVENT fulfills its main promise. Using PREVENT the total number of SLO violations decreases to about 28% of the number of predicted violations. However, we can also see that PREVENT does not primarily prevent violations, but rather aims at minimizing the costs of violations. For instance, for SLO 4 and 5 the total number of violations even increases. This is because these SLOs are conflicting with the first SLOs, and SLO 1 is in general the most expensive one to violate. Hence, PREVENT happily trades violations of SLO 4 and 5 for preventing violations of SLO 1. Thereby, the total costs for the service provider can be reduced to 56% of the predicted costs. The lower part of the table validates the claim of the paper that it makes sense to incorporate the costs of adaptation into the decision pro-

	SLO 1	SLO 2	SLO 3	SLO 4	SLO 5	Total
Considering Costs						
Violations Predicted/Actual	209/129	442/1	390/42	75/104	0/41	1116/317 (28.4%)
Avg. Costs Predicted/Actual	5207/2904	884/2	6264/558	840/1068	0/9	14923/8415 (56.4%)
Ignoring Costs						
Violations Predicted/Actual	223/40	449/0	245/17	66/218	0/115	983/390 (39.7%)
Avg. Costs Predicted/Actual	5521/756	898/0	4086/216	756/2364	0/29	12632/22241 (176.1%)

TABLE 6: End-to-End Results

cess. To that end, we have modified the target function of the optimization in such a way that the costs of adaption are ignored. In this configuration, the total costs after adaptation are 176% of the predicted costs. That means that in this experiment it is in fact much more expensive for the provider to prevent adaptations (in the way that optimization ignoring costs suggests) than doing nothing at all.

7 Related Work

To the best of our knowledge, no approaches with the exact focus of this paper (cost-based optimization of service compositions) have been published so far. However, there are some areas relevant or related to this problem, which we discuss in the following.

On a fundamental level, our work is based on the notion that both atomic and composite services exhibit some measurable quality (QoS). Monitoring QoS has been an active research area for some time. Different techniques proposed in this direction include monitoring based on client feedback [25], monitoring of TCP-level metrics using network analysis techniques [15] or event-based monitoring based on event-condition-action rules [26]. We use the VRESCo event engine and event-based monitoring in a manner very similar to the approach presented in [26].

The PREvent approach aims at autonomous optimization of service compositions with regards to SLA violations and costs of adaptation. This bears a natural resemblance to the idea of QoS optimization for service compositions, as prominently described in [27]. Later approaches tried to improve on this concept by using more efficient heuristic algorithms, e.g., H1_RELAX_IP [28] (a heuristic relaxation of integer programming), WFlow [29] (based on stochastic workflow reduction) or the immune algorithm [30]. Different authors approached the problem by combining global optimization and local selection (which can be done much more efficient than global optimization). This approach can also be considered a heuristic, since the combination with local selection does not guarantee a globally optimal solution [31]. Most comparably to our work, the authors of [32] use a genetic algorithm combined with local search to efficiently solve the QoS optimization problem. The main difference of our work to all these approaches is that we do not optimize the composition with regard to global QoS goals. Instead, our optimization goal is to minimize the costs resulting from SLA violations and adaptations. Therefore, in our work, some SLAs are allowed to be violated if it is financially desirable for the provider to do so. Hence, the optimization problem we have to solve is different.

To our work, even more important than the measurement of past quality is the prediction of future QoS. One well-known approach to establishing predictable QoS levels in a composite service is QoS aggregation, i.e., the process of calculating the quality dimensions of a composite service based on the QoS of the utilized services and aggregation functions. QoS aggregation has for instance been discussed in [20]. The concept of QoS aggregation has been extended to SLA aggregation by several authors [33], [34]. As an alternative to QoS and SLA aggregation, different authors have proposed to use various machine learning techniques to predict composition QoS from monitored runtime data [6], [7]. This approach is also the one that we use in PREvent, as explained in Section 3. The main advantage that we see in using machine learning is that it is very easy to incorporate non-QoS data (composition instance data, such as customer identifiers or ordered products) without the need to explicitly specify aggregation rules describing how this data influences the composition performance. However, note that the contribution discussed in this paper is in principle agnostic of the actual approach used for prediction of violations.

Generally, PREvent is a system to monitor and prevent SLA violations. In this area, some works exist, which discuss the runtime monitoring of composition quality, such as [35]. This paper is of particular interest to us, since it discusses an integrated approach towards monitoring based on events. As stated above, this is quite related to monitoring in PREvent. These works do not attempt to explain the reasons for SLA violations, and neither do they try to prevent them. The MoDe4SLA approach [4] is a top-down approach towards identifying these influential factors of SLA violations. Research in a similar direction, but using data mining techniques instead of top-down analysis, has also been presented in [5]. Our work is different in that we do not only try to identify which parts of a service composition cause SLA violations, but actively prevent them by applying targeted adaptation actions. Therefore, our system essentially implements the paradigm of self-adapting service compositions. This is related to the area of flexible service composition, as introduced in [36]. Flexible service compositions reoptimize their composition at runtime, in order to deal with unanticipated problems. Similar ideas (self-healing processes) have also been presented as part

of the DISC framework [37], which implements dynamic and only partially defined processes. A different kind of self-healing processes have been discussed in [38]. In this paper, the authors present the VieDAME framework, which autonomously monitors the QoS of services used in the composition, and triggers service re-selection if the monitored QoS falls below a given threshold. This is similar to the PREVENT approach, but our system supports a wider range of adaptation actions (as discussed in our earlier work [8]). Additionally, [38] does not take the costs of adaptation into account. Another middleware for self-adapting compositions is MASC [39]. However, the authors of this paper focus more on adaptation for functional reasons, while our main goal is the optimization of non-functional aspects. Furthermore, the MASC system also does not explicitly incorporate costs of adaptation.

The core contribution of this paper is the notion that there generally is a tradeoff to consider between preventing SLA violations and the costs of doing so. Hence, a composite service provider is maximizing his own revenue by minimizing his total costs. Similar models have been investigated in many related areas before. For instance, [40] describes a model for revenue maximizing in Web services hosting using dynamic admission policies. Similarly, techniques to optimize application servers in a way to maximize the provider profit in distributed systems have been proposed in [41]. Other tradeoffs that have been discussed in the literature include the performance-security tradeoff [42] or the tradeoff between composition QoS and the costs of monitoring [43].

8 Conclusion and Future Work

For providers of composite Web services, it is essential to be able to minimize cases of SLA violations. One possible route to achieve this is to predict at runtime, which instances are in danger of violating SLAs, and to apply various adaptation actions to these instances only. However, it is not trivial to identify which adaptations are the most cost-effective way to prevent any violation, or if it is at all possible to prevent a violation in a cost-effective way. In this paper, we have modelled this problem as a one-dimensional, discrete optimization problem. Furthermore, we have presented both, deterministic and heuristic solution algorithms. We have evaluated these algorithms based on a manufacturing case study, and shown which types of algorithms are better suited for which scenarios.

The main current limitation is that adaptation is only considered on instance level, that is, for each composition instance separately. Aggregate SLOs, which are defined over a number of instances, are out of scope. Similarly, at the moment we do not consider 'permanent' adaptations, i.e., adaptations which are done for all future instances. We believe that the PREVENT adaptation model can be extended to this kind of SLOs and actions, but new approaches to predict violations and impact models are needed to this end.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube) and 257483 (Indenica).

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, November 2007.

[2] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's Inside the Cloud? An Architectural Map of the Cloud Landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–31.

[3] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web Services on Demand: WSLA-Driven Automated Management," *IBM Systems Journal*, vol. 43, no. 1, pp. 136–158, January 2004.

[4] L. Bodenstaff, A. Wombacher, M. Reichert, and M. C. Jaeger, "Analyzing Impact Factors on Composite Services," in *Proceedings of the 2009 IEEE International Conference on Services Computing (SCC '09)*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 218–226.

[5] B. Wetzstein, P. Leitner, F. Rosenberg, S. Dustdar, and F. Leymann, "Identifying Influential Factors of Business Process Performance Using Dependency Analysis," *Enterprise Information Systems*, vol. 4, no. 3, pp. 1–8, July 2010.

[6] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime Prediction of Service Level Agreement Violations for Composite Services," in *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, 2009, pp. 176–186.

[7] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, "Event-Driven Quality of Service Prediction," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 147–161.

[8] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'10)*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 369–376.

[9] P. Leitner, B. Wetzstein, D. Karastoyanova, W. Hummer, S. Dustdar, and F. Leymann, "Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'10)*, 2010.

[10] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of Service-Based Applications Based on Process Quality Factor Analysis," in *Proceedings of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, 2009, pp. 395–404.

[11] "Business Process Modeling Notation Specification," Object Management Group (OMG), Tech. Rep., 2006.

[12] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, pp. 1–42, May 2009.

[13] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo," *IEEE Transactions on Services Computing*, vol. 3, pp. 193–205, July 2010.

[14] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[15] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 205–212.

[16] S. Haykin, *Neural Networks and Learning Machines: A Comprehensive Foundation*, 3rd ed. Prentice Hall, 2008.

[17] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81–106, March 1986.

[18] D. Ivanovic, M. Carro, and M. Hermenegildo, "Towards Data-Aware QoS-driven Adaptation for Service Orchestrations," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 107–114. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2010.73

[19] L. Juszczyk and S. Dustdar, "Script-Based Generation of Dynamic Testbeds for SOA," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 195–202. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2010.75

[20] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS Aggregation for Web Service Composition using Workflow Patterns," in *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 149–159.

[21] L. Xu and B. Jennings, "A Cost-Minimizing Service Composition Selection Algorithm Supporting Time-Sensitive Discounts," in *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 402–408.

[22] T. Feo and M. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[23] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[24] N. Radcliffe and P. Surry, "Formal Memetic Algorithms," *Evolutionary Computing*, vol. 865, pp. 1–16, 1994.

[25] R. Jurca, B. Faltings, and W. Binder, "Reliable QoS Monitoring Based on Client Feedback," in *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. New York, NY, USA: ACM, 2007, pp. 1003–1012.

[26] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 132–144.

[27] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[28] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," in *Proceedings of the 2006 IEEE International Conference on Web Services (ICWS'06)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 72–82.

[29] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web*, vol. 1, May 2007.

[30] J. Xu and S. Reiff-Marganiec, "Towards Heuristic Web Services Composition Using Immune Algorithm," in *Proceedings of the 2008 IEEE International Conference on Web Services (ICWS'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 238–245.

[31] M. Alrifai and T. Risse, "Combining Global Optimization With Local Selection for Efficient QoS-Aware Service Composition," in *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. New York, NY, USA: ACM, 2009, pp. 881–890.

[32] F. Rosenberg, M. B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions," in *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC'10)*.

[33] T. Unger, F. Leymann, S. Mauchart, and T. Scheibler, "Aggregation of Service Level Agreements in the Context of Business Processes," in *Proceedings of the 12th International Enterprise Distributed Object Computing Conference (EDOC'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 43–52.

[34] I. Haq, A. Huqqani, and E. Schikuta, "Aggregating Hierarchical Service Level Agreements in Business Value Networks," in *Proceedings of the 7th International Conference on Business Process Management (BPM'09)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 176–192.

[35] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + Astro: An Integrated Approach for BPEL Monitoring," in *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 230–237.

[36] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

[37] E. Zahoor, O. Perrin, and C. Godart, "DISC: A Declarative Framework for Self-Healing Web Services Composition," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 25–33.

[38] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL," in *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. New York, NY, USA: ACM, 2008, pp. 815–824.

[39] A. Erradi, P. Maheshwari, and V. Tosic, "Policy-Driven Middleware for Self-Adaptation of Web Services Compositions," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (Middleware'06)*. New York, NY, USA: Springer-Verlag New York, Inc., 2006, pp. 62–80.

[40] M. Mazzucco, I. Mitrani, J. Palmer, M. Fisher, and P. McKee, "Web Service Hosting and Revenue Maximization," in *Proceedings of the Fifth European Conference on Web Services (ECOWS'07)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 45–54.

[41] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning Servers in the Application Tier for E-Commerce Systems," *ACM Transactions on Internet Technology*, vol. 7, no. 1, p. 7, 2007.

[42] S. S. Yau, Y. Yin, and H. G. An, "An Adaptive Tradeoff Model for Service Performance and Security in Service-Based Systems," in *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 287–294.

[43] Y. Zhang, M. Panahi, and K.-J. Lin, "Service Process Composition with QoS and Monitoring Agent Cost Parameters," in *Proceedings of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 311–316.

Philipp Leitner has a BSc and MSc in business informatics from Vienna University of Technology. He is currently a PhD candidate and university assistant at the Distributed Systems Group at the same university. Philipp's research is focused on middleware for distributed systems, especially for SOAP-based and RESTful Web services.

Waldemar Hummer holds a BSc (Univ. of Innsbruck) and MSc (Vienna Univ. of Technology) in Computer Science, and a BSc in Business Administration (WU Vienna). Currently he is a university assistant and PhD candidate at the Distributed Systems Group, Vienna Univ. of Technology. His main topics of interest are in the area of self-optimizing service-based systems, Web service composition and Web data aggregation.

Schahram Dustdar is Full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Vienna University of Technology (TU Wien). He is also Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands.

Towards Optimizing the Non-Functional Service Matchmaking Time

Kyriakos Kritikos
ICS-FORTH
Heraklion, Crete, Greece
kritikos@ics.forth.gr

Dimitris Plexousakis
ICS-FORTH
Heraklion, Crete, Greece
dp@ics.forth.gr

ABSTRACT

The Internet is moving fast to a new era where million of services and things will be available. In this way, as there will be many functionally-equivalent services for a specific user task, the service non-functional aspect should be considered for filtering and choosing the appropriate services. The related approaches in service discovery mainly concentrate on exploiting constraint solving techniques for inferring if the user non-functional requirements are satisfied by the service nonfunctional capabilities. However, as the matchmaking time is proportional to the number of non-functional service descriptions, these approaches fail to fulfill the user request in a timely manner. To this end, two alternative techniques for improving the non-functional service matchmaking time have been developed. The first one is generic as it can handle non-functional service specifications containing n-ary constraints, while the second is only applicable to unary-constrained specifications. Both techniques were experimentally evaluated. The preliminary evaluation results show that the service matchmaking time is significantly improved without compromising matchmaking accuracy.

Keywords

service, discovery, matchmaking, non-functional, QoS, optimization, performance, constraint programming

1. INTRODUCTION

The Web is now moving to a new era, called the Web of Services (WoS), where millions of services and things will be available to users through a converged information, communication, and service infrastructure. Service-orientation enables such a change, as it is one of the main mechanisms exploited by application developers to build added-value applications through the discovery and composition of services.

However, service-orientation has not yet delivered its promise as it relies on the role of the service broker, which is not successfully fulfilled by current realizations. Such realizations have failed due to the following factors. First, they are not scalable to handle thousands or millions of service requests and advertisements. Second, they exhibit low matchmaking performance and accuracy. Third, they have mainly concentrated on the functional aspect and have neglected the non-functional one. However, as there will be many functionally-equivalent services for a specific user task, the non-functional aspect should be considered for filtering and selecting the best among these services.

Concerning the functional aspect, many matchmaking approaches have been developed [1, 4], mainly exploiting Semantic Web techniques and focusing solely on the service I/O. Such approaches exhibit good performance but their accuracy is not perfect. This is justified by the fact that the service functionality in terms of its pre- and post-conditions is not considered, partly because the service specifications (advertisements and requests) do not include the description of such an aspect. Due to the move to the WoS, specific approaches [5, 3] have concentrated on scalability issues and on further optimizing the service matchmaking time by appropriately organizing the service advertisement and request space. Such approaches move to the right direction but still face the challenge of imperfect accuracy.

While the research status concerning functional service discovery is satisfactory, the same cannot be stated for non-functional service discovery. The state-of-the-art approaches [2] in the latter sub-area exhibit perfect accuracy and mainly focus on optimizing the time required to matchmake a single non-functional request-to-advertisement pair by exploiting appropriate constraint solving techniques. However, they have not yet focused on optimizing the overall non-functional matchmaking time. As such, they can take significant time to match a set of hundreds or thousands of non-functional service advertisements against a non-functional service request, so they are not yet appropriate for the move to the IoS era. In addition to the above problem, there have not been approaches focusing on scalability issues.

Thus, there is a need for scalable techniques that are able to appropriately manage a vast amount of non-functional service offers and optimize their matchmaking time. If such techniques were coupled with those exploited for functional service discovery, then a better service broker realization would be offered, enabling users to discover those services matching their tasks both functionally and non-functionally in a timely manner and with the appropriate accuracy.

Two novel techniques were developed to close this research gap that significantly optimize the non-functional service matching time with respect to the techniques already proposed.

WWW '12 Lyon, France

2. PROPOSED NON-FUNCTIONAL SERVICE MATCHING TECHNIQUES & EVALUATION

The first proposed technique, called "Subsumes matching" technique, relies on the "subsumes" type of matchmaking metrics and on the fact that if a non-functional service specification A subsumes another specification B then it will also subsume all the specifications subsumed by B. To this end, it organizes the non-functional service advertisement space in such a way that the number of non-functional request-to-advertisements pairs examined is less than that the state-of-the-art approach in [2]. In particular, it constructs a forest of "subsumes" trees, where each node corresponds to a non-functional service advertisement and a parent node in each tree subsumes all of its children nodes. In this way, when a service request is issued, it is compared against the nodes of each tree from the root until the leaves. However, if it is found that it subsumes a specific node, then there is no need to go further down to the node's children/descendants as the request will certainly match/subsume them.

This technique is obviously quicker than the state-of-the-art one, as each one uses the same matchmaking metric but the first technique performs less comparisons. However, the construction and update of a forest of "subsumes" trees is more costly than the construction and the update of a list of service advertisements (as it is the case for the state-of-the-art approach). To this end, this technique exhibits a higher insertion, deletion, and update time with respect to the state-of-the-art approach. This technique can be distributed by assigning the responsibility of matching a subset of the subsumes trees to different nodes.

The second proposed technique (i.e., the "Unary matching" one) relies on similar techniques performed in functional service matchmaking [1] in order to appropriately organize the non-functional service advertisement space. In particular, it maintains for each non-functional metric/property an ordered set of limits, where each limit may correspond to one or more non-functional specifications containing a respective metric bound or equality constraint on the limit's value. To this end, when a non-functional service request is issued, each of its unary constraints are examined based on their containing metric. Depending on the metric bound and constraint type, a sub-part of the metric's ordered list of limits is examined so as to produce a list of the matching non-functional advertisements' URIs. For instance, if the request constraint is of the form: $X \leq a$, then the limits that are equal or less to a are examined and the URIs of the non-functional specifications that have constraints of the form $X \leq a_1$, or $X == a_1$, where $a_1 \leq a$, are collected. For each request constraint, its constructed URI list is concatenated with that of the previous constraint. If the URI list concatenation is empty, then the non-functional request does not have a matching advertisement. Otherwise, after the processing of the last request constraint, the final, concatenated URI list contains all the URIs of the advertisements that match the request.

The second technique is quicker than the other proposed technique as well as the prominent matchmaking approach not only in terms of matchmaking but also insertion, deletion, and update time. Moreover, due to the way it organizes the advertisement space, it can be easily distributed by assigning the responsibility of matching a sub-set of all non-functional metrics to different nodes. Its sole drawback is that it relies on exploiting only unary-constrained non-functional service specifications.

Both techniques were experimentally evaluated against one state-of-the-art technique. The preliminary results show that both techniques exhibit a better matchmaking time than that of the state-of-the-art one and that the second technique is significantly better and more scalable than the others.

3. CONCLUSION

This paper has proposed two novel techniques for optimizing the non-functional service matchmaking time. Moreover, it has reported some initial preliminary evaluation results which indicate that both techniques are significantly better than the state-of-the-art one and that the second proposed technique is more scalable.

Concerning future work, as there is a trade-off in optimizing both matchmaking and offer registration performance, research must concentrate on discovering the exact point where the performance in both aspects is satisfactory. In addition, future research should concentrate on further experimenting with the proposed techniques and appropriately integrating such techniques in functional service brokers. Moreover, additional techniques can be considered which intelligently organize also the service demand space. Finally, scalable and distributed, functional and non-functional service discovery mechanisms must be developed incorporating the two novel non-functional service matchmaking techniques. In this way, if the latter mechanisms are exploited by a service broker, then the vision of the Internet of Services will come closer to its realization.

4. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

5. REFERENCES

[1] O. Cliffe and D. Andreou. Service Matchmaking Framework. Public Deliverable D5.2a, Alive EU Project Consortium, 10 September 2009. Available at: `http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=28&Itemid=49`.

[2] K. Kritikos and D. Plexousakis. Mixed-Integer Programming for QoS-Based Web Service Matchmaking. *IEEE Trans. Serv. Comput.*, 2(2):122–139, 2009.

[3] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *J. Syst. Softw.*, 81(5):785–808, 2008.

[4] P. Plebani and B. Pernici. URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642, 2009.

[5] M. Stollberg, M. Hepp, and J. Hoffmann. A Caching Mechanism for Semantic Web Service Discovery. In *ICWS*, 2007.

Cost Reduction Through SLA-driven Self-Management

André Lage Freitas, Nikos Parlavantzas, Jean-Louis Pazat
Université Européenne de Bretagne
INSA, INRIA, IRISA, UMR 6074
F-35708 Rennes, France
Emails: {Andre.Lage,Nikos.Parlavantzas,Jean-Louis.Pazat}@irisa.fr

***Abstract*—A main challenge for service providers is managing service-level agreements (SLAs) with their customers while satisfying their business objectives, such as maximizing profits. Most current systems fail to consider business objectives and thus to provide a complete SLA management solution. This work proposes an SLA-driven management solution that aims to maximize the provider's profit by reducing resource costs as well as fines owning to SLA violations. Specifically, this work proposes a framework that comprises multiple, configurable control loops and supports automatically adjusting service configurations and resource usage in order to maintain SLAs in the most cost-effective way. The framework targets services implemented on top of large-scale distributed infrastructures, such as clouds. Experimental results demonstrate its effectiveness in maintaining SLAs while reducing provider costs.**

I. Introduction

Service-based systems are built by integrating loosely-coupled services from a range of providers. To handle varying service loads, providers are increasingly taking advantage of large-scale distributed infrastructures, such as clouds and grids, which deliver remote resources in a flexible, on-demand fashion. A major challenge for service providers is managing such infrastructures in order to meet their business objectives while maintaining conformance to Service-Level Agreements (SLAs) with customers.

A large part of the research on SLA management in service-oriented architectures (SOAs) targets composite services; that is, services composed of simpler services, and thus shielded from the details of the underlying infrastructure. SLA management in this context typically involves replacing services by more suitable ones [10], [22]. Such work does not address how basic, atomic services guarantee Quality of Services (QoS) properties, which invariably requires managing the underlying distributed infrastructure, and is the focus of this paper. A significant amount of work has focused on SLA management for large-scale distributed applications, such as e-science applications deployed on grids, or multi-tier enterprise applications deployed on clusters [9], [4], [12]. However, such work does not address meeting the business objectives of service providers, such as maximizing profit.

This paper proposes a generic framework to assist service providers in honoring SLAs while reducing the costs of infrastructure usage. The proposed framework integrates a rich set of QoS management mechanisms supporting the complete SLA life-cycle, from SLA template creation to service termination. To manage infrastructure usage, the framework builds on a simple interface, compatible with modern grid and IaaS cloud APIs. To accommodate fluctuating service loads and unpredictable faults, the framework includes flexible support for self-adaptation in the form of multiple interacting control loops. Importantly, the control loops build on replaceable adaptation strategies, which can be combined in multiple ways, thus extending the applicability of the framework.

The rest of the paper is structured as follows. Section II introduces the Qu4DS (Quality Assurance for Distributed Services) framework while its adaptation strategies are exposed in Section III. Section IV discusses some aspects of service provisioning, how Qu4DS profiles service providers and the assumptions on which it relies. Next, Section V presents implementation details along with the flac2ogg service provider. Section VI discusses the evaluation, its environment and the results. Finally, related work is commented in Section VII, followed by the conclusion in Section VIII.

II. Qu4DS

A. Architecture

The main goal of Qu4DS is to provide SLA management that minimizes the service provider's costs. Costs include the price of using the underlying infrastructure and the payment of fines due to SLA violations. Specifically, Qu4DS divides a pool of booked resources among distinct SLA contracts and uses these resources to execute service requests based on agreed QoS properties. In order to handle dynamic events, such as resource shortages and faults, Qu4DS performs management actions guided by configurable strategies, which attempt to minimize costs. For example, to deal with resource shortages, Qu4DS chooses the most suitable request to abort in order to minimize fine payment.

The Qu4DS architecture is described by Figure 1. It is located at the PaaS layer and uses resources from an IaaS provider in order to provide support to the upper SaaS layer. In the SaaS layer, customers contact the Web Service (WS) in order to establish a contract. The contract proposal is forwarded to the *SLA Negotiator* that asks the *QoS Translator* to translate its QoS to the resource configuration able to ensure such QoS. The SLA Negotiator then checks through the *infrastructure management* interface whether the resource requirements can

be met. If so, the SLA Negotiator configures and deploys the service instance on the infrastructure through the *job management* interface and confirms the contract agreement to the right customer which is now able to send requests.

When a customer sends a request, the SLA Negotiator asks the *request arrival* control loop whether it can be treated. If so, the SLA Negotiator forwards the request to the right service instance deployed on the infrastructure. The service instance prepares the distributed tasks necessary to treat the request and asks Qu4DS to execute the tasks. These tasks are also deployed by Qu4DS on the infrastructure through the job management interface and monitored by the *job failure* and *job delay* control loops. If any of the control loops informs the SLA Negotiator that the request could not be treated, the SLA Negotiator aborts the request, informs the customer of the SLA violation and computes the penalties.

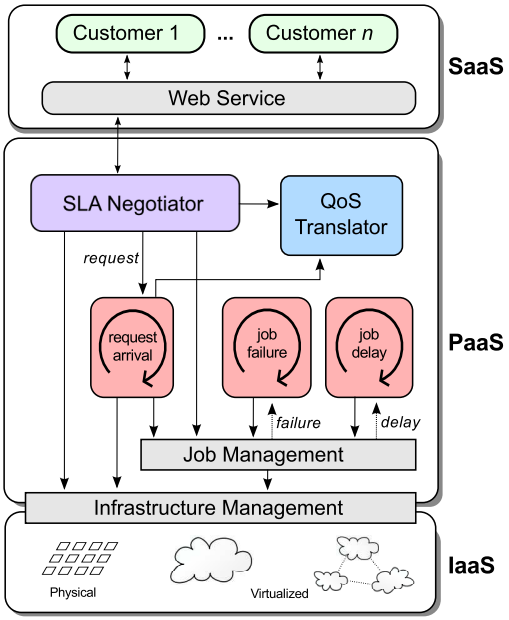


Fig. 1. Qu4DS Architecture.

B. Interfaces

To define the interactions between Qu4DS and underlying infrastructures, we have investigated several distributed infrastructure interfaces [11], [6], [7], [8]. Grid interfaces, such as the SAGA [8] interface, define a complex but complete set of functionalities based on the job abstraction. The main limitation of such interfaces is that they mix together two distinct concerns, job management and resource acquisition. On the other hand, current IaaS cloud providers provide decoupled, simple resource provisioning interfaces [1], [20], [3], [16]. However, these interfaces are limited to the provisioning of low-level resources, typically virtual machines, with no support for high-level abstractions, such as jobs.

Therefore, we have structured the interaction among Qu4DS, underlying infrastructures and customers around the three interfaces depicted in Table I. First, the *infrastructure management* interface exposes a set of operations to reserve resources of specific types. This interface can be implemented on top of existing IaaS cloud APIs, which provide operations to create and manage virtual machine (VM) instances. Second, since we believe that jobs are a useful, high-level abstraction for managing service instances, we defined a *job management* interface based on SAGA. This interface extends the SAGA job life-cycle (cf. Figure 2) and enables creating, canceling, suspending as well as migrating jobs between resources. Finally, we defined the *Web Service* interface at the highest level through which contracts are negotiated and customers send requests. The idea here is to enable the provider to negotiate contracts rather than to define a complete protocol for SLA negotiation, such as [17], [2]. The negotiation process requires that the customer obtains contract templates from the provider which is in charge of auditing the provision and monitoring penalties. Further methods for service provisioning may be added to this interface according to provider requirements.

Infrastructure Management
getResourceTypes()
getNumberOfAvailableResources(resourceClass)
reserve(nOfResources, resourceClass, startTime, endTime)
resize(resourceClass, newNumberOfResources, endTime, resourcesId)
getReservedResources(reservationId)
Job Management
createJob(jobDescription)
runJob(job, resourceAddress)
cancelJob(job)
cancelAllJobsOnResource(resource)
checkpointJob(job)
suspendJob(job)
resumeJob(job)
resumeJob(job,checkpointVersion)
migrateJob(job, resourceAddress)
getAllJobs(reservationId)
getJobs(jobState)
getJobsOnResource(resourceAddress)
registerCallback(job, metric, activate, observer)
Web Service
getListOfContractTemplates()
contractEstablishment(contractTemplate)
request(args)

TABLE I
DESCRIPTION OF THE INTERFACES USED BY QU4DS ENTITIES.

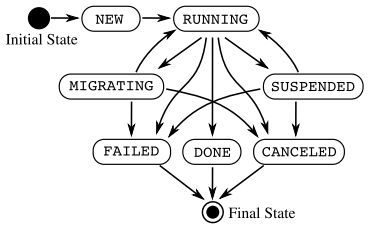


Fig. 2. The job life-cycle extended from the SAGA Task life-cycle.

III. Self-Adaptation

The application of adaptation mechanisms depends on the Qu4DS request life-cycle, which is described by Figure 3. When a customer demands the service, a request is created containing information about its SLA specifications and the request state is set to `NEW`. If the request cannot be treated, the state is set to `ABORTED`. Otherwise, the state is set to `TREATING` until the end of the treatment. If the request is successfully treated, it is then considered as `TREATED`, otherwise it is `ABORTED`. In addition, during request treatment, Qu4DS registers information about aborted requests in order to subsequently compute the penalties.

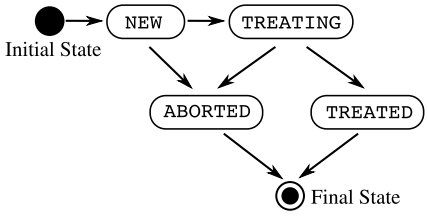


Fig. 3. Qu4DS request life-cycle.

Qu4DS applies the Autonomic Computing [13] concept by implementing three MAPE (Monitor, Analyze, Plan, Execute) control loops as shown in Figure 1. The first control loop, the *request arrival* loop, is driven by request arrival events and is in charge of checking whether just-arrived, `NEW` requests can be treated. Specifically, the loop asks the QoS Translator to translate the QoS to resource requirements and then checks resource availability with the infrastructure. If the resource requirements cannot be met, the control loop decides if it will abort this request or another request that is currently being treated. Furthermore, the loop aborts a request to let another be executed only if the resource requirements of the former satisfy the resource requirements of the latter.

The other control loops ensure the proper execution of requests and operate over `TREATING` requests. Indeed, these control loops act as self-healing mechanisms in order to prevent SLA violations. While the *job failure* control loop reacts to job failure events by providing reliable request treatment, the *job delay* control loop reacts to job delays and ensures performance aspects of the request treatment.

Furthermore, all control loops take decisions based on the adaptation strategies with which each loop is configured. We have currenltly designed the adaptation strategies described in Table II. These strategies are driven by the high-level goal of minimizing the provider costs of fine payments, thus maximizing the provider's profit.

Request Arrival Strategies	
Name	**Description**
VFC (Violation based on Fine Cost)	Aborts the request whose fine is the cheapest.
VBP (Violation Based on Priority)	Aborts the request with lowest priority.
RVC (Random Violation Criterion)	Random choice of request to be aborted.
Job Failure Strategies	
Name	**Description**
FJV (Failed Job implies Violation)	Aborts the request immediately.
SRFJ (Single Re-Submission of Failed Jobs)	Replaces the failed job once. If the same job is failed again, it aborts the request.
Job Delay Strategies	
Name	**Description**
LJV (Late Job implies Violation)	Aborts the request immediately.
SRLJ (Single Replacement of Late Jobs)	Replaces the delayed job once. If the same job gets late again, it aborts the request.

TABLE II
Qu4DS adaptation strategies are triggered by the arrival of new requests, the occurrence of job failures and delays.

IV. Cost-Reducing Service Provisioning

A. Service-Level Agreement

We compiled a basic SLA template based on common aspects of current SLA specifications [2], [17], as exposed in the following.

- Parties
  - Service Customer
  - Service Provider
- Terms
  - Description: *the service functional requirements*
  - Duration: t seconds
  - Customer's obligations: should not exceed the maximum request frequency ϕ_{max}, i.e., the maximum number of requests per t.
  - Provider's obligations: should provide the agreed service according to its QoS.
  - SLA type: varies among *platinum, gold, silver*
  - Priority: varies among *low, normal, high*
  - Price model: pay-per-use (usage-dependent)
  - Price: p_i
- QoS
  - Throughput (MB/sec): th

The SLA types (*platinum, gold, silver*) are differentiated by the throughput that they allow. The SLA template includes the contract priority, which means that higher-priority customers are preferred against lower-priority customers in case of demand overload. Regarding the pricing model, customers pay depending on the duration of using the service in a usage-dependent fashion [14], [15]. This pricing model was chosen because infrastructure costs are typically based on the same model. The provider can thus adjust the price of the service based on infrastructure costs. Specifically, the total price of a contract is given by $p_i = p_{type}(t) + p_{priority}$ where $p_{type}(t)$ is the price per time for each SLA type and $p_{priority}$ is the price of the chosen priority. Both prices are defined by the provider, which fills the SLA templates. These templates are then selected by customers according to the suitable SLA type and priority.

B. Profiling

Qu4DS relies on profiling the service provider by executing requests with different resource configurations. During profiling, the QoS Translator collects the response time threshold ($rp_{threshold}$) to process a given amount of data d and the job execution time threshold ($j_{threshold}$). This data is then used to calculate QoS values and further information for checking adaptation strategies conditions. First, the QoS Translator calculates the standard deviations for both thresholds rp_{st_dv} and j_{st_dv} respectively. Then it calculates the job execution time threshold $j_{threshold}$ using the following equation, where e_1 and e_2 are adjusting coefficients:

$$j_{threshold} = e_1 \times j_{mean} + e_2 \times j_{st\_dv} \quad (1)$$

Following that, the QoS Translator calculates the request response time threshold $rp_{threshold}$ which defines the maximum amount of time that request treatment can take. If the request elapsed time r_{etime} is greater than $rp_{threshold}$, then the request is aborted implying an SLA violation. In Equation 2, $rp_{threshold}$ is defined, where $ad_{overhead}$ is a fixed time that represents the overhead of employing adaptation actions.

$$rp_{threshold} = rp_{mean} + rp_{st\_dv} + ad_{overhead} + j_{threshold} \quad (2)$$

Subsequently, the adaptation threshold $ad_{threshold}$ is calculated, which is used to decide whether there is enough time to trigger an adaptation action:

$$ad_{threshold} = rp_{mean} + rp_{st\_dv} \quad (3)$$

Finally, the QoS Translator calculates the QoS throughput th based on the following equation, where d is the size of the profiling data:

$$th = \frac{d}{rp_{threshold}} \quad (4)$$

C. Assumptions

To make the service provisioning model more concrete, let us rely on the following assumptions and equations.

Assumption 1: The service provider's profit is given by the difference between its total revenue and total costs of fine payments and infrastructure utilization.

Based on Assumption 1, the Equation 5 defines the service provider's profit P_t given an operational time t. The $\sum_{i=0}^{n_t} p_i$ is the sum of the provider's revenue from all agreed contracts, where n_t is the total number of contracts and p_i is the total revenue of a contract (cf. Section IV-A). Let us define the set F_t as the set of all fines during t where $f_k \in F$; hence $\sum_{k=1}^{|F_t|} f_k$ is the sum of all the fines the service provider has to pay during t. Finally, $c_t \times a_t$ is the total cost for all booked resources during t where c_t is the price for using a single resource during t and a_t is the total amount of booked resources.

$$P_t = \sum_{i=0}^{n_t} p_i - \sum_{k=1}^{|F_t|} f_k - c_t \times a_t \quad (5)$$

Assumption 2: If all requests are violated, the service provider will make zero profit.

The Equation 6 calculates the maximum frequency a customer can reach in terms of number of requests during the contract duration t for contract i :

$$\phi_{max_i} = \frac{t}{rp_{i_{threshold}}} \quad (6)$$

Based on Assumption 2 and supposing that customers perform the maximum feasible number of requests (ϕ_{max_i}) per t, we deduce from Equation 5 the fine value of the contract i where r_i is the number of resources required by i:

$$f_i = \frac{p_i - c_t \times r_i}{\phi_{max_i}} \quad (7)$$

Assumption 3: The service provider wants to share the resources among distinct contracts in order to reduce the costs of resource acquisition.

Thus, let us assume that service instances are deployed on dedicated resources and their associated distributed tasks are deployed on shared resources in order to save costs. In particular, we assume that this sharing leads to a $g\%$ decrease in the amount of needed resources. The following equation defines a as the total number of acquired resources by the service provider, where n is the total number of contracts, $type$ is the SLA type index (0 means silver, 1 gold and 2 platinum), u_{type} is the number of contracts of $type$, and w_{type} is the number of distributed tasks that $type$ requires:

$$a = n + \frac{(100 - g)}{100} \times \sum_{type=0}^{2} u_{type} \times w_{type} \quad (8)$$

V. Implementation and Case Study

A. Qu4DS Components

We have implemented the *infrastructure management* interface (cf. Section II-B) using Grid'5000 [5] as the IaaS layer. A customized Grid'5000 image was created [1] that contains all programs and libraries required to execute Qu4DS, including an implementation of the job management interface. A similar image would be used in an Amazon EC2-based implementation.

The implementation of the *job management* interface (cf. Section II-B) follows a layered design. The higher layer deals with job life-cycle state management (cf. Figure 2) and keeps Qu4DS informed about job metrics following the publish/subscribe pattern. In this layer, raw job metrics (e.g., UNIX process metrics) are mapped to high-level job metrics and job states. The middle layer abstracts over the use of different backend batch systems, implemented at the lower

[1]The details of this image can be accessed here: https://www.grid5000.fr/mediawiki/index.php/Lenny-x64-quads

layer. Currently, two backend implementations are available: one that supports the XtreemOS grid [6] and another on top of SSH (Secure Shell). In addition, the job management implementation is able to simulate job misbehavior by randomly choosing jobs that will present failures or delays.

Concerning the implementation of the *control loops*, they rely on an event-condition-action decision engine. The adaptation strategies that will guide them are loaded from the Qu4DS configuration file and applied at runtime. When events are received, the control loops check the strategy conditions and decide whether to trigger adaptations. While the request arrivals control loop always reacts to the arrival of new requests, the other control loops check more specific conditions. For instance, if the SRFJ adaptation strategy is enabled (cf. Table II), the job failure control loop checks if there is enough time to adapt ($r_{etime} < ad_{threshold}$, cf. Equation 3) and whether the misbehaving job is already a job replacement (since replacements should not be replaced).

B. Case Study: flac2ogg Service Provider

The current Qu4DS implementation targets distributed service providers based on the Master/Worker paradigm. In this context, the service instance represents the master, and the distributed tasks represent the workers. As a case study, we have implemented the *flac2ogg* service provider that converts FLAC [23] audio files to OGG [24] files, as illustrated by Figure 4. Based on SLA templates, customers establish contracts with the service provider by choosing the desired throughput th QoS (MB/secs) and the priority. The QoS translation gives the amount of resources necessary to satisfy a given throughput by mens of number of workers (w). Next, the SLA Negotiator configures the master with the right number of workers that can provide the required throughput and deploys the master on the infrastructure. When a request sent by a customer reaches the master through the SLA Negotiator, the master splits the FLAC file in w parts, prepares w workers for encoding each part, and asks Qu4DS to execute and manage the workers. Qu4DS wraps the workers as jobs and submits them to be executed in parallel through the job management interface. During worker execution, the control loops may react to job metric-related events triggering adaptation actions. When the workers finish encoding the FLAC parts, Qu4DS provides the results to the master, which merges them and calls the SLA Negotiator. The SLA Negotiator finally forwards the OGG file to the right customer.

VI. Evaluation

An evaluation was performed in order to study the effectiveness of Qu4DS in performing SLA-driven self-management. The following paragraphs discuss how Qu4DS was calibrated, the scenarios, the testbed and the obtained results.

A. Calibrating Qu4DS to flac2ogg

We profiled the flac2ogg service provider based on a 194MB FLAC file which means approximatively thirty minutes of recorded audio. Table III shows the QoS table together with

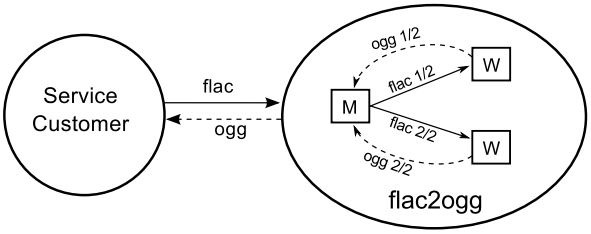


Fig. 4. The flac2ogg service provider encodes FLAC audio files to OGG based on the Master/Worker pattern. In this example, the number of worker w is set to two.

$rp_{i_{threshold}}$ and ϕ_{max_i} (cf. Equations 2 and 6) where $e_1 = 1.5$ and $e_2 = 3$ (cf. Equation 1).

SLA Type i	**Throughput (MB/sec)**	**Resource Configuration**	$rp_{i_{threshold}}$ **(secs)**	ϕ_{max_i} **(#req/**t**)**
Silver	1.49	2 workers	130	11
Gold	1.90	3 workers	102	15
Platinum	2.34	6 workers	83	19

TABLE III
QU4DS QOS TABLE CALIBRATED BASED ON A 194MB FLAC FILE. THE THROUGHPUT WAS CALCULATED BASED ON $rp_{i_{threshold}}$.

Figure 5 represents the customers' demand by means of a request schedule for contract duration $t = 1800$ seconds. The beginning of the bars represents the start time of each request and the end represents $rp_{i_{threshold}}$. The Y-axis represents the IDs of customers' contracts, whose total number is fifteen. Their priorities were chosen randomly; the bolder the line is, the higher priority it has. With respect to SLA types, the first five contracts hold silver SLA type, the next five gold, and the last five platinum. Note that even though the end of some requests may overlap the beginning of others, this does not necessarily mean that they will be running at the same time. For instance, if a request is executed normally with no need to adapt, it will probably finish in time $rp_{mean} + rp_{st_dv}$ and not in time $rp_{i_{threshold}}$. Moreover, although the mean of requests per hour was set to 75% of ϕ_{max}, the actual request frequency ϕ_i for each customer i was obtained from a Poisson number generator, which explains why the total request numbers of contracts of the same SLA type are not the same. Additionally, we decreased by two the value of ϕ_{max} (cf. Equation 6) to introduce some spare time, which was convenient for our practical scenarios.

B. Scenarios and Testbed

The evaluation scenarios were defined based on the combinations of adaptation strategies shown in Table IV. The amount of jobs that were configured to be failed was 10% and 10% for delayed jobs. Further, we assume that a job could not be failed and late simultaneously which means that eighteen jobs were failed as well as other eighteen were delayed. Regarding the delayed jobs, they never stopped processing which led the request elapsed time reach $rp_{threshold}$ thus triggering adaptation actions in A scenarios. With respect to job failures,

job crash times were randomly chosen during their execution. Additionally, all the contracts were established before starting the request scheduling to ensure that the time to establish the contracts would not compromise the request scheduling punctuality.

Strategy	VFC	VBP	RVC
SRFJ and SRLJ	A1	A2	A3
FJV and LJV	B1	B2	B3

TABLE IV
THE EVALUATION SCENARIOS WERE DEFINED BY COMBINING THE ADAPTATION STRATEGIES.

Regarding the testbed, the experiments were performed on Grid'5000 resources which have the same characteristics: 8-core 2.5 GHz CPU, 32GB RAM computers interconnected through a Gigabit network connection. In order to set the number of booked resources, Qu4DS relied on Equation 8 setting $g = 50$ resulting in $a = 43$ booked resources. Moreover, we assumed that the cost of using a resource was $c_{1800} = 0.05$ €.

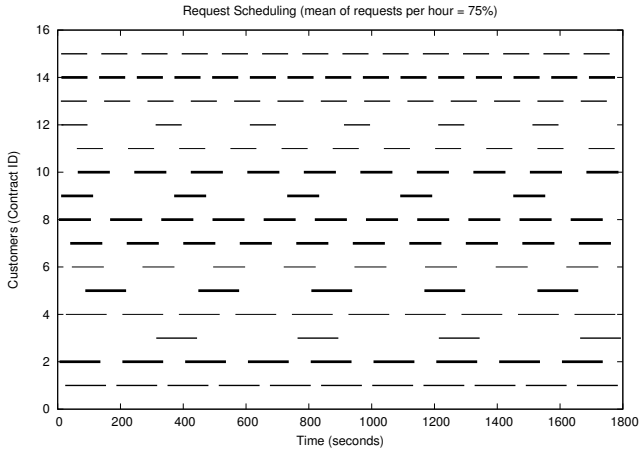


Fig. 5. The request scheduling took into account a mean of $\phi = 75\%$ of ϕ_{max}. Contract priorities were chosen randomly.

C. Results

Figures 6, 7 and 8 show the results of the aforementioned evaluation scenarios. In Figures 6 and 7, each cluster represents a sub-scenario on which is plotted the mean of the percentage of `TREATED` and `ABORTED` requests for each contract ID. In these figures, the mean of the profit is also plotted for each sub-scenario. Figure 8 shows a box-and-whisker diagram for the profit values; note that the maximum profit a scenario could reach (i.e., with no SLA violations) is $P_{1800} = 15.225$. Furthermore, each experiment was repeated five times.

When analyzing the results, the first important aspect to observe is the effectiveness of the SRFJ and SRLJ strategies in reducing the number of SLA violations. This is evident in the greater percentage of aborted requests on Scenario B (cf. Figs. 6 and 7) as well as their lower profit values if compared to Scenario A (cf. Fig. 8). This was an expected behavior since FJV and LJV automatically abort requests when either a job failure or delay occurs in order to free the resources before reaching the request response time threshold.

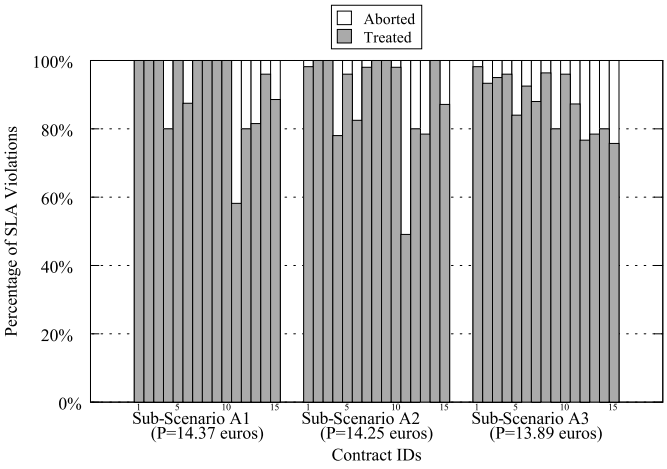


Fig. 6. Results of Scenario A: SRFJ and SRLJ activated.

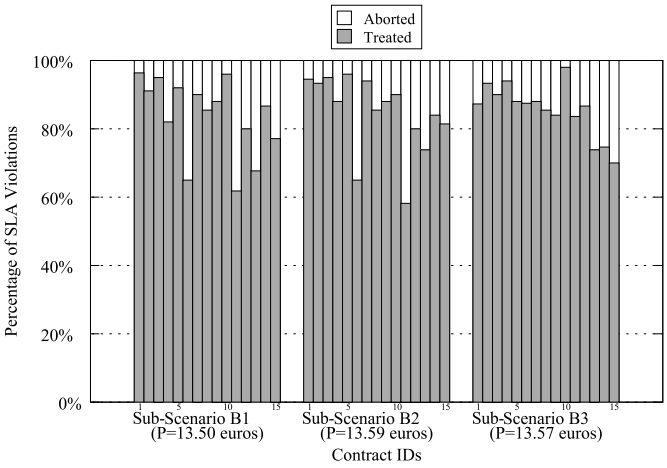


Fig. 7. Results of Scenario B: FJV and LJV activated.

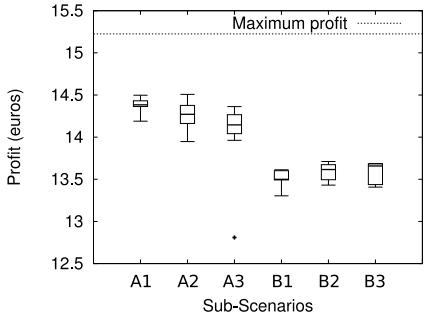


Fig. 8. Box-and-whisker diagram of profit values.

Interestingly, the results show an unexpected behavior in Scenario B: the RVC strategy was more efficient than anticipated. It guaranteed greater profit than VFC and practically the same profit as VBP (cf. Fig. 8). This occurred because no adaptation strategy considers future consequences on request scheduling when taking decisions. Indeed, the strategies do not rely on predicting customers' demand. Thus, random choices of which request should be aborted might fit better the request scheduling, as observed in Scenario B. This unexpected behavior leads to the observation that VFC and VBP adaptation strategies are efficient if combined to others that handle job failures and delays such as the configuration of Scenario A.

Finally, there was no significant difference among the profit values of the same scenario even though different percentages of requests were aborted. This is because fine values (cf. Equation 7) varied weakly among 0.05 and 0.09 € approximatively. Moreover, the amount of requests ($\phi = 75\%$ of ϕ_{max}) was relatively low for the reduced amount of shared resources ($g = 50$); besides, the requests were well dispersed and thus there were no rush moments on the request scheduling.

VII. Related Work

Service-oriented computing has widely investigated how services can be discovered, composed, monitored and managed in order to guarantee the proper execution of service-based applications. With regards to SLA management, specifications such as WS-Agreements [2] and WSLA [17] specify how agreements can be negotiated and provide guidelines for monitoring and auditing service behaviors. Much research work addresses specific aspects of SLA management, such as application on resource management [9], SLA enforcement [12] and integrated SLA management [25]. In [21], the author proposes hierarchical SLA management that enforces SLA on top of distinct adaptation policies; the policies adjust network traffic based on current QoS values, guided by high-level objectives.

While all aforementioned approaches address QoS assurance, they do not provide techniques aiming specifically at reducing costs. In this context, in [18], [19] the authors propose to maximize the profit of cloud IaaS providers by means of resource over-provisioning and dynamically setting the price according to the supply and demand. Some virtual machines (VMs) are chosen to be shut down to enable resource over-provisioning. Although this approach targets profit maximization, it relies on the availability QoS which is measured as VM up-time and SLA violation means the time VMs had been shut down. Moreover, the solution is placed on the IaaS layer thus not directly supporting the development of SaaS service providers as Qu4DS supports.

The SLA@SOI project [25] addresses a similar problem with this work, but in a different way. Specifically, SLA@SOI proposes an integrated architecture for SLA management that associates SLAs with multiple elements of the software stack at multiple layers. On the other hand, Qu4DS addresses SLA management at a single (PaaS) layer. Furthermore, the SLA@SOI project provides a set of highly generic building blocks, intended to be applicable to arbitrary deployment contexts. Qu4DS provides a complete management solution for web-service providers that build on utility infrastructures, while allowing extensibility with respect to adaptation strategies and infrastructure technologies.

VIII. Conclusions

This paper has presented a framework, Qu4DS, that facilitates SLA management for services built on distributed infrastructures, such as IaaS clouds. The framework contributes to increasing the provider profit by dynamically managing services and resources, taking into account SLA prices, fines, and infrastructure costs. The framework is modularly structured as a set of control loops configured with replaceable adaptation strategies, thus increasing its applicability to different application domains, workload characteristics, and adaptation objectives. The framework includes mechanisms for SLA negotiation and QoS translation, thus supporting in an integrated way the full SLA life-cycle, from contract negotiation to service termination. The paper has also presented detailed experimental results demonstrating that the framework can effectively increase provider profits and maintain SLA compliance in dynamic environments.

We intend to continue this work in several ways. First, we intend to add support for dynamically adjusting the number of booked resources to match current demand and to avoid over-provisioning, taking full advantage of the elastic capabilities of cloud infrastructures. Second, we intend to develop additional adaptation strategies and to evaluate them in the context of various workload and infrastructure conditions. Supporting the automated selection of suitable adaptation strategies is a longer-term goal of this work.

IX. Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-CUBE). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, April 2011.

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, 2007.

[3] M.-E. Bégin. An EGEE Comparative Study: Grids and Clouds - Evolution or Revolution. Technical report, CERN - Engeneering and Equipment Data Management Service, June 2008.

[4] S. Benkner and G. Engelbrecht. A Generic QoS Infrastructure for Grid Web Services. *Advanced International Conference on Telecommunications / Internet and Web Applications and Services, International Conference on*, 0:141, 2006.

[5] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[6] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld. XtreemOS: a Vision for a Grid Operating System. Technical report, XtreemOS Consortium, May 2008.

[7] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21:513–520, 2006.

[8] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. Saga: A simple api for grid applications - high-level application programming on the grid. *Computational Methods in Science and Technology: special issue "Grid Applications: New Challenges for Computational Methods"*, SC05:8(2), Nov. 2005.

[9] P. Hasselmeyer, B. Koller, L. Schubert, and P. Wieder. Towards SLA-Supported Resource Management. In *HPCC '06: Proceedings of the 2006 International Conference on High Performance Computing and Communications*, pages 743–752. Springer, 2006.

[10] F. Irmert, T. Fischer, and K. Meyer-Wegener. Runtime adaptation in a service-oriented component model. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 97–104, New York, NY, USA, 2008. ACM.

[11] S. Jha, A. Merzky, and G. Fox. Using Clouds to Provide Grids with Higher Levels of Abstraction and Explicit Support for Usage Modes. *Concurrency and Computation: Practice & Experience*, 21:1087–1108, 2009.

[12] Jose Antonio Parejo and Pablo Fernandez and Antonio Ruiz-Cortés and José María García. SLAWs: Towards a Conceptual Architecture for SLA Enforcement. In *Services, IEEE Congress on*, volume 0, pages 322–328. IEEE Computer Society, 2008.

[13] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[14] S. Lehmann and P. Buxmann. Pricing Strategies of Software Vendors. *Business & Information Systems Engineering*, 1(6):452–462, 2009.

[15] S. Lehmann, T. Draisbach, P. Buxmann, and C. Koll. Pricing Models of Software as a Service Providers: Usage-Dependent Versus Usage-Independent Pricing Models. In G. Dhillon, editor, *Proceedings of the 8th Annual Conference on Information Science, Technology & Management (CISTM'10)*, August 2010.

[16] Lillard, Terrence V. and Garrison, Clint P. and Schiller, Craig A. and Steele, James. *The Future of Cloud Computing*, pages 319–339. Elsevier, 2010.

[17] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification. Technical report, IBM, 2003.

[18] M. Macíandas, J. Fitó and, and J. Guitart. Rule-based SLA management for revenue maximisation in Cloud Computing Markets. In *CNSM'10: Proceedings of the 6th IEEE/IFIP International Conference on Network and Service Management*, pages 354 –357, oct. 2010.

[19] M. Macíandas and J. Guitart. Maximising Revenue in Cloud Computing Markets by means of Economically Enhanced SLA Management. Technical Report UPC-DAC-RR-2010-32, Universitat Politècnica de Catalunya. Departament d'Arquitectura de Computadors, 2010.

[20] Nurmi, Daniel and Wolski, Rich and Grzegorczyk, Chris and Obertelli, Graziano and Soman, Sunil and Youseff, Lamia and Zagorodnov, Dmitrii. The Eucalyptus Open-Source Cloud-Computing System. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[21] P. R. Pereira. Service Level Agreement Enforcement for Differentiated Services. In *Second International Workshop of the EURO-NGI Network of Excellence*, Villa Vigoni, Italy, July 13-15 2005.

[22] S-CUBE Consortium. Taxonomy of Adaptation Principles and Mechanisms. Deliverable 1.2.2, May 2009.

[23] The FLAC project. Free Lossless Audio Codec (FLAC). http://flac.sourceforge.net/, 2011.

[24] The Xith Open Source Community. The Ogg container format. http://xiph.org/ogg/, July 2010.

[25] W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, J. Lambea. A Reference Architecture for Multi-Level SLA Management. *Journal of Internet Engineering*, 4, 2010.

Autonomic SLA-aware Service Virtualization for Distributed Systems

Attila Kertész, Gábor Kecskeméti
Computer and Automation Research Institute
MTA SZTAKI
H-1518 Budapest, P.O. Box 63, Hungary
attila.kertesz, kecskemeti@sztaki.hu

Ivona Brandic
Distributed Systems Group
Vienna University of Technology
1040 Vienna, Argentinierstr. 8/181-1, Austria
ivona@infosys.tuwien.ac.at

***Abstract*—Cloud Computing builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Managing such heterogeneous environments requires sophisticated interoperation of adaptive coordinating components. In this paper we introduce an SLA-aware Service Virtualization architecture that provides non-functional guarantees in the form of Service Level Agreements and consists of a three-layered infrastructure including agreement negotiation, service brokering and on demand deployment. In order to avoid costly SLA violations, flexible and adaptive SLA attainment strategies are used with a failure propagation approach. We demonstrate the advantages of our proposed solution with a biochemical case study in a Cloud simulation environment.**

***Keywords*-Cloud Computing; SLA-negotiation; Service Brokering; On-demand deployment;**

I. Introduction

Cloud Computing [6] builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Both Grids and Service Based Applications (SBAs) already provide solutions for executing complex user tasks, but they are still lacking non-functional guarantees. The newly emerging demands of users and researchers call for expanding service models with business-oriented utilization (agreement handling) and support for human-provided and computation-intensive services [6]. Providing guarantees in the form of Service Level Agreements (SLAs) are highly studied in Grid Computing [18]. Nevertheless in Clouds, infrastructures are also represented as a service that are not only used but also installed, deployed or replicated with the help of virtualization. These services can appear in complex business processes, which further complicates the fulfillment of SLAs in Clouds. For example, due to changing components, workload and external conditions, hardware and software failures, already established SLAs may be violated. Frequent user interactions with the system during SLA negotiation and service executions (which are usually necessary in case of failures), might turn out to be an obstacle for the success of Cloud Computing. Thus, there is demand for the development of SLA-aware Cloud middleware, and application of appropriate strategies for autonomic SLA attainment. Despite cloud computing's business-orientation, the applicability of Service-level agreements in the Cloud middleware is rarely studied, yet [23]. Most of the existing work address provision of SLA guarantees to the consumer and not necessarily the SLA-based management of loosely coupled Cloud infrastructure. In such systems it is hard to localize where the failures have happen exactly, what the reason is for the failure and which reaction should be taken to solve the problem. Such systems are implemented in a proprietary way, making it almost impossible to exchange the components (e.g. use another version of the broker).

Autonomic Computing is one of the candidate technologies for the implementation of SLA attainment strategies. Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [10]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems (e.g. human body) autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware and software failures. An important characteristic of an autonomic system is an intelligent closed loop of control. The Autonomic Manager (AM) manages the element's state and behaviour. It is able to sense state changes of the managed resources and to invoke appropriate set of actions to maintain some desired system state. Typically control loops are implemented as MAPE (monitoring, analysis, planning, and execution) functions [10]. The *monitor* collects state information and prepares it for the *analysis*. If deviations to the desired state are discovered during the analysis, the *planner* elaborates change plans, which are passed to the *executor*. However, for the successful implementation of the autonomic principles loosely coupled SLA-based Cloud middleware is required. In such system failure source should be identified based on violated SLAs and located exactly considering different components of a Cloud middleware (virtualization, brokering, negotiation, etc. components). Thus, once the failure is identified Service Level Objectives (SLOs) can be used as a guideline for the

autonomic reactions.

In this paper we introduce a novel architecture considering resource provision using a virtualization approach and combining it with the business-oriented utilization used for the SLA agreement handling. Thus, we provide an SLA-coupled infrastructure for on-demand service provision based on SLAs (called SLA-based Service Virtualization – SSV). We also exemplify how autonomic behaviour appears in the architecture in order to cope with changing user requirements and on demand failure handling. The main contributions of this paper include: (i) the presentation of the novel loosely coupled architecture for the *SLA-based Service virtualization* and on-demand resource provision, (ii) description of the architecture including *meta-negotiation, meta-brokering, brokering and automatic service deployment* with respect to its autonomic behaviour, and (iii) the *validation* of the SSV architecture with a biochemical case study in a Cloud simulation environment.

II. Related work

Though Cloud Computing is highly studied and a large body of work has been done trying to define and envision the boundaries of this new area, despite business-orientation, the applicability of Service-level agreements in the Cloud middleware is rarely studied, yet [23]. The envisioned framework in [7] proposes a solution to extend the web service model by introducing and using semantic web services. The need for SLA handling, brokering and deployment also appears in this vision, but they focus on using ontology and knowledge-based approaches. Most of related works consider either virtualization approaches [8] without taking care of agreements or concentrate on SLA management neglecting the appropriate resource virtualizations [21]. The work by Van et. al. [22] studied the applicability of the autonomic computing to Cloud-like systems, but they almost exclusively focus on virtualization issues like Virtual Machine (VM) packing and placement.

Comparing the currently available solutions, autonomic principles are not implemented in a adequate way because of they are lacking an SLA-coupled Cloud infrastructure, where failures and malfunctions can be identified using well defined SLA contracts. Work presented in this paper is the first attempt to develop an SLA-coupled autonomic Cloud infrastructure in an adequate way. Works presented in [19], [17] discuss incorporation of SLA-based resource brokering into existing Grid systems, but they do not deal with virtualization. The Rudder framework [15] facilitates automatic Grid service composition based on semantic service discovery and space based computing. Venugopal et al. propose a negotiation mechanism for advance resource reservation using the alternate offers protocol [26], however, it is assumed that both partners understand the alternate offers protocol.

Current deployment solutions do not leverage their benefits on higher level. For example the Workspace Service (WS) [8] as a Globus incubator project supports wide range of scenarios involving virtual workspaces, virtual clusters and service deployment from installing a large service stack to deploy a single WSRF service if the Virtual Machine image of the service is available. It is designed to support several virtual machines. The XenoServer open platform [20] is an open distributed architecture based on the XEN virtualization technique aiming at global public computing. The platform provides services for server lookup, registry, distributed storage and a widely available virtualization server. Also the VMPlants [14] project proposes an automated virtual machine configuration and creation service which is heavily dependent on software dependency graphs, but this project stays within cluster boundaries.

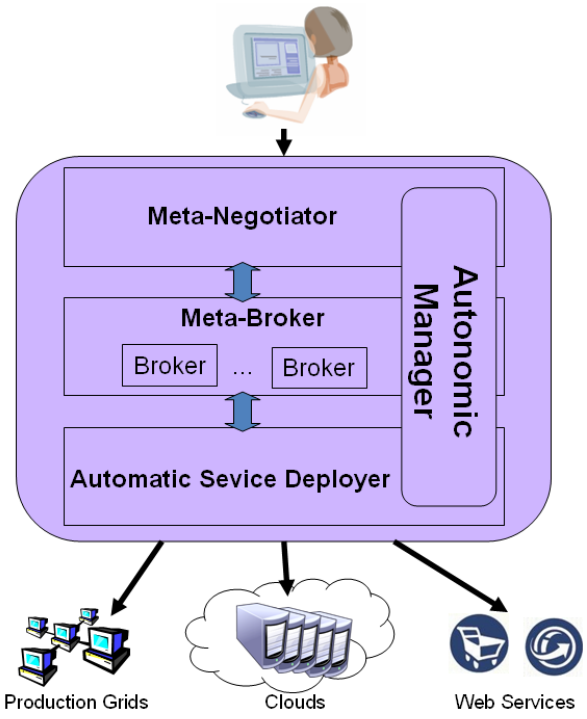


Figure 1. The SSV architecture.

III. The autonomic, SLA-based Service virtualization (SSV) architecture

In this section we present a unified service architecture that builds on three main areas: agreement negotiation, brokering and service deployment using virtualization (more detailed descriptions on the architecture can be read in [11]). We suppose that service providers and service consumers meet on demand and usually do not know about the negotiation protocols, document languages or required infrastructure of the potential partners. Figure 1 shows our

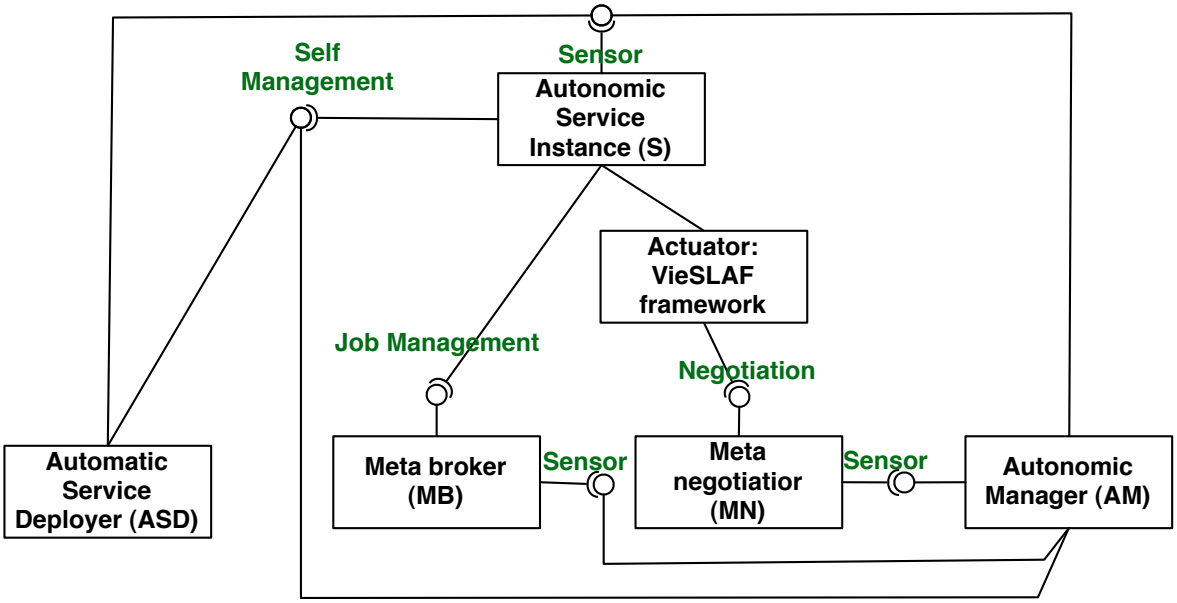


Figure 2. Autonomic components in the SSV.

proposed, general architecture. The main components of the architecture and their roles are gathered in Table I, while the sequence diagram in Figure 3 reveals the interactions of the components and the utilization steps of the architecture:

Agreement negotiation. First the user starts a negotiation for executing a service with certain QoS requirements (specified in a Service Description (SD) with an SLA). Then the Meta-Negotiator (MN) asks the Meta-Broker (MB), if it could execute the service with the specified requirements. Later on MB matches the requirements to the properties of the available brokers and replies with an acceptance or a different offer for renegotiation. After this phase the MN replies with the answer of MB. These initial steps may continue for renegotiations, until both sides agree on the terms.

Service request. After the agreement has been established the user calls the service with the SD and SLA. Then the MN passes SD and the transformed SLA (to the protocol the selected broker (B) understands) to the MB. In the next phase the MB calls the selected Broker with SLA and a possibly translated SD (to the language of the Broker). Then the broker executes the service with respect to the terms of the SLA (if needed deploys the service before execution). Finally the result of the execution is reported to the Broker, the MB, the MN, finally to the User (or workflow engine).

Information collection. While serving requests the architecture also processes background tasks that are not in the ideal execution path, these are also presented on Figure 3. The following background tasks are information collecting procedures that provide accurate information about the current state of the infrastructure up to the meta-broker level. In *"step a"* the Automatic Service Deployer (ASD) monitors the states of the virtual resources and deployed services. Then in *"step b"* ASD reports service availability and properties to its Broker. Finally at *"step c"* all Brokers report available service properties to the MB.

A. Autonomic behaviour in the SSV architecture

The sequence diagram of Figure 3 represents the ideal execution flow of the SSV architecture, therefore it does not reflect cases when unexpected events occur during the operation of each component. The SSV architecture targets these events with the autonomic management interfaces introduced on Figure 2. We distinguish three types of interfaces in our architecture: the *job management interface*, the *negotiation interface* and the *self-management interface*. Negotiation interfaces are typically used by the monitoring processes of brokers and meta-broker during the negotiation phases of the service deployment process. Self-management is needed to re-negotiate established SLAs during service execution. Job management interfaces are necessary for the manipulation of services during execution, for example for the upload of input data, or for the download of output data, and for starting or canceling service executions.

The *Autonomic manager* in the SSV architecture is an abstract component that specifies how self-management is carried out. The manager can be implemented in the different layers of the architecture in order to handle the different unexpected situations. For the identification of failures we use case-based reasoning with a knowledge database, since the identification of failure sources (done by sensors) and mapping failures to appropriate reactions (done by actuators)

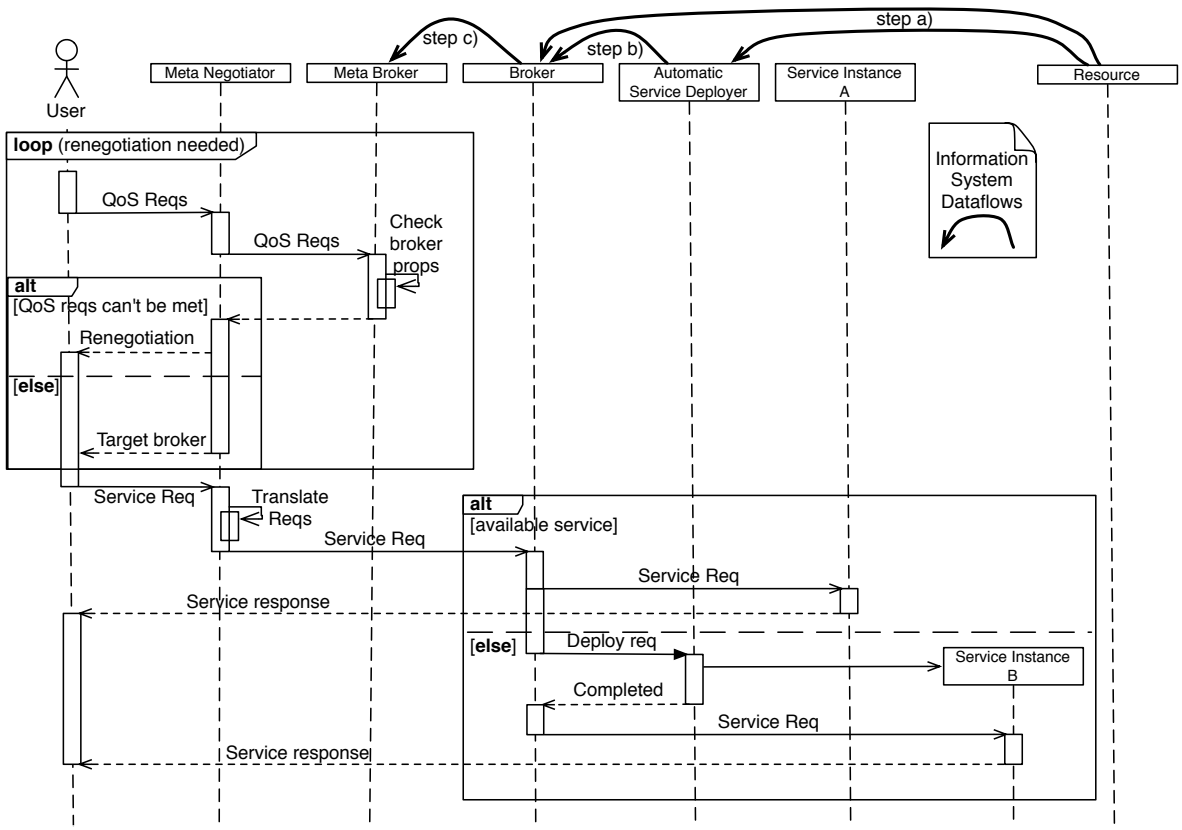


Figure 3. Component interactions in the SSV architecture.

are trivial tasks, and they are part of our ongoing work [1]. However the discussion of the knowledge database and layered notifications is out of scope of this paper, therefore here we mention three typical reactive actions we support in our SSV: namely *negotiation bootstrapping*, *broker breakdown* and *self-initiated deployment*. Negotiation bootstrapping occurs when the architecture needs to select a new negotiation strategy. Broker breakdowns are handled by the Meta-Broker component by initiating renegotiation on an already executed service call. Finally a service instance can guarantee the negotiated SLAs by deploying another service instance and redirecting the calls causing the unexpected service behaviour (more details on these failures and reactive actions can be read in [12]). An example software actuator used for the meta-negotiations is the VieSLAF framework [3], which bridges between the incompatible SLA templates by executing the predefined SLA mappings.

B. Dependencies between the SSV components

During the negotiation process the MB interacts with MN: it receives a service request with the service description and SLA terms, and looks for a deployed service reachable by some broker that is able to fulfil the specified terms. If a service is found, the SLA will be accepted and the and MN notified, otherwise the SLA will be rejected. If the service requirements are matched and only the terms cannot be fulfilled, it could continue the negotiation by modifying the terms and wait for user approval or further modifications.

The Information Collector (IC) component of MB stores the data of the reachable brokers and historical data of the previous submissions. It also communicates with the ASDs, and receives up-to-date data about the available services and predicted invocation times (that might also include service deployment before the actual invocation). This information shows whether the chosen broker is available, or how reliable its services are.

Finally service brokers could instruct ASDs to *deploy* a new service instance. However, deployments could also occur independently from the brokers as explained in the following. After these deployments the ASD has to *notify* the corresponding service brokers about the infrastructure changes. This notification is required, because the IC caches the state of the SBA for scalability. Thus even though a service has just been deployed on a new site, the broker will not direct service requests there. This is especially needed when the deployment was initiated to avoid an SLA violation (e.g. self-initiated deployment).

Table I
ROLES IN SSV ARCHITECTURE

Acronym	Role	Description
U	User	A person, who wants to use a service
MN	Meta-Negotiator	A component that manages Service-level agreements. It mediates between the user and the Meta-Broker, selects appropriate protocols for agreements; negotiates SLA creation, handles fulfilment and violation ([2]).
MB	Meta-Broker	Its role is to select a broker that is capable of executing/deploying a service with the specified user requirements ([13]).
B	Broker	It interacts with virtual or physical resources, and in case the required service needs to be deployed it interacts directly with the ASD.
ASD	Automatic Service Deployment	It installs the required service on the selected resource on demand ([9]).
S	Service	The service that users want to deploy and/or execute
R	Resource	Physical machines, on which virtual machines can be deployed/installed.

IV. EVALUATION OF THE SSV ARCHITECTURE WITH CLOUDSIM

In order to evaluate our proposed SSV solution, we use a typical biochemical application as a case study called TINKER Conformer Generator application [24], gridified and tested on production Grids. The application generates conformers by unconstrained molecular dynamics at high temperature to overcome conformational bias then finishes each conformer by simulated annealing and energy minimization to obtain reliable structures. Its aim is to obtain conformation ensembles to be evaluated by multivariate statistical modeling.

The execution of the application consists of three phases: The first one is performed by a generator service responsible for the generation of input data for parameter studies (PS) in the next phase. The second phase consist of a PS subworkflow, in which three PS services are defined for executing three different algorithms (dynamics, minimization and simulated annealing – we refer to these services and the generator service as TINKERALG), and an additional PS task that collects the outputs of the three threads and compresses them (COLL). Finally in the third phase, a collector service gathers the output files of the PS subworkflows and uploads them in a single compressed file to the remote storage (UPLOAD). These phases contain 6 services, out of which four are parameter study tasks that are executed 50 times. Therefore the execution of the whole workflow means 202 service calls. We set up the simulation environment for executing a similar workflow.

For the evaluation, we have created a general simulation environment, in which all stages of service execution in the SSV architecture can be simulated and coordinated. We have created the simulation environment with the help of the CloudSim toolkit [4] (that includes and extends GridSim). It supports modeling and simulation of large scale Cloud computing infrastructure, including data centers, service brokers and provide scheduling and allocations policies. Our general simulation architecture that builds both on GridSim and on CloudSim, can be seen in Figure 4. On the left-bottom part we can see the GridSim components used for the simulated Grid infrastructures, and on the right-bottom part we can find CloudSim components. Grid resources can be defined with different Grid types, they consist of more machines, to which workloads can be set, while Cloud resources are organized into Datacenters, on which Virtual machines can be deployed. Here service requests are called as cloudlets, which can be executed on virtual machines. On top of this simulated Grid and Cloud infrastructures we can set up brokers. Grid brokers can be connected to one or more resources, on which they can execute so-called gridlets (ie. service requests). Different properties can be set to these brokers and various scheduling policies can also be defined. A Cloud broker can be connected to a data center with one or more virtual machines, and it is able to create and destroy virtual machines during simulation, and execute cloudlets on these virtual machines. The Simulator class is a CloudSim entity that can generate a requested number of service requests with different properties, start and run time. It is connected to the created brokers and able to submit these requests to them (so is acts as a user or workflow engine). It is also connected to the Grid Meta-Broker Service through its web service interface and able to call its matchmaking service for broker selection.

We submitted the simulated workflow in three phases: in the first round 61 service requests for input generation, then 90 for executing various TINKER algorithms, finally in the third round 51 calls for output preparation. The simulation environment was set up similarly to the real Grid environment we used for testing the TINKER workflow application. Estimating the real sizes of these distributed environments, we set up four simulated Grids (GILDA, VOCE, SEEGRID and BIOMED [5]) with 2, 4, 6 and 8 resources (each of them had 4 machines). Out of the 202 jobs 151 had special requirements: they use the TINKER library available in the last three Grids, which means these calls need to be submitted to these environments, or to Cloud resources (with pre-deployed TINKER environments). The simulated execution time of the 150 parameter study services were set to 30 minutes, the first generator service to 90

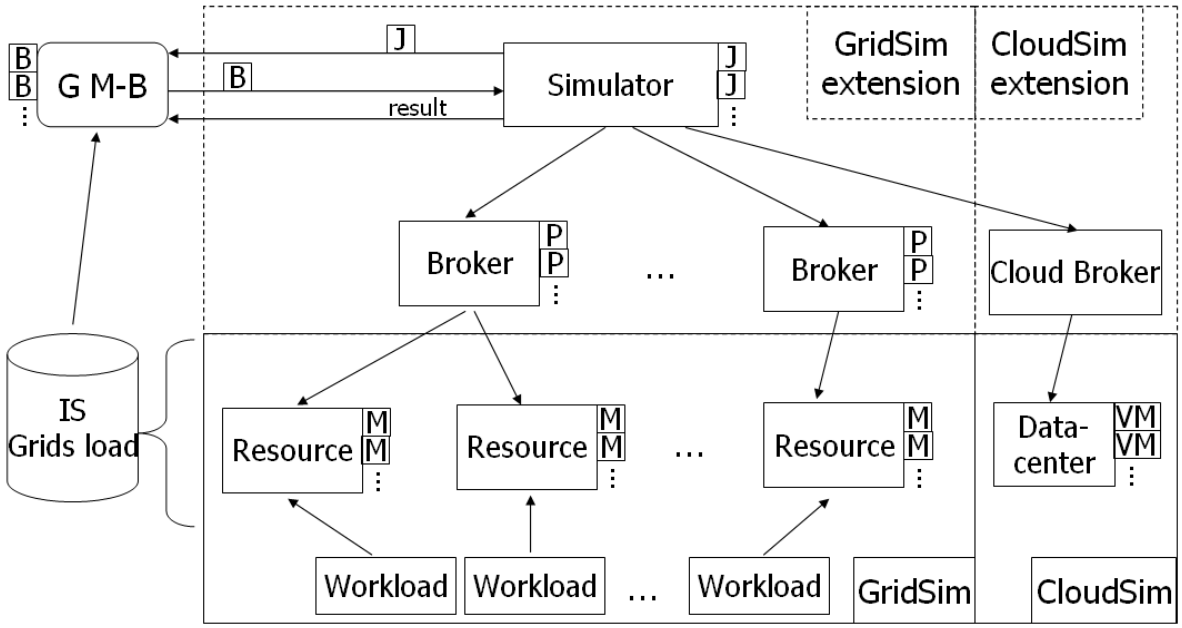


Figure 4. Simulation architecture with CloudSim.

minutes, and the other 51 were set to 10 minutes. All of the four brokers (set to each simulated Grid one-by-one) used random resource selection policy, and all the resources had background workload, for which the traces were taken from the Grid Workloads Archive (GWA) [25] (we used the GWA-T-11 LCG Grid log file). In our simulation architecture we used 20 nodes (called resources in the simulation), therefore we partitioned the logs and created 20 workload files (out of the possible 170 according to the number of nodes in the log). The sorting of the job data to files from the original log file were done continuously, and their arrival times have not been modified, and the run time of the jobs also remained the same. According to these workload files the load of the simulated environments are shown in Figure 5 (which are also similar to the load experienced on the real Grids). One Cloud broker has also been set up. It managed four virtual machines deployed on a data center with four hosts of dual-core CPUs. In each simulation all the jobs were sent to the Meta-Broker to select an available broker for submission. It takes into account the actual background load and the previous performance results of the brokers for selection. If the selected Grid broker had a background load that exceeded a predefined threshold value, it selected the Cloud broker instead.

Out of the 202 workflow services 150 use TINKER binaries (three different algorithms are executed 50 times). These requirements can be formulated in SLA terms, therefore each service of the workflow has an SLA request. If one of these requests are sent to the Cloud broker, it has to check if a virtual machine (VM) has already been created that is able to fulfil this request. If there is no such VM, it deploys one on-the-fly. For the evaluation we used three different Cloud broker configurations: in the first one four pre-deployed VMs are used – one for each TINKER algorithm (TINKERALG) and one for data collecting (capable for both COLL and UPLOAD, used by the last 51 jobs). In the second case we used only one pre-deployed VM, and deployed the rest on-the-fly, when the first call arrived with an SLA. Finally in the third case, when a VM received more then 15 requests the Cloud broker duplicated it (in order to minimize the overall execution time).

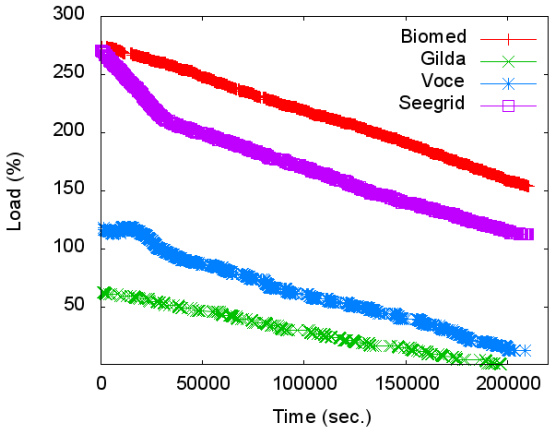


Figure 5. Workload of simulated Grids.

Regarding on-demand deployment, we have created 4 virtual appliances encapsulating the four different services

our TINKER workflow is based on (namely TINKERALG, COLL and UPLOAD – we defined them in the beginning of this section). Then we have reduced the size of the created appliances with ASD's virtual appliance optimization facility. Finally we have deployed each service 50 times on an 8 node (32 CPU) Eucalyptus [16] cluster, and measured the interval between the deployment request and the service's first availability. Table II shows the measurement results for the TINKERALG, COLL and UPLOAD images. These latencies were also applied in the simulation environment within the Cloud broker.

Table II
DEPLOYMENT TIMES OF THE DIFFERENT SERVICES IN THE TINKER APPLICATION.

Service	Average deployment time	Standard deviation
GEN	8.2 sec	1.34 sec
TINKERALG	8.3 sec	1.48 sec
COLL	6.9 sec	0.84 sec
UPLOAD	6.9 sec	1.21 sec

In order to evaluate the performance of our proposed SSV solution we compare it to a general meta-brokering architecture used in Grid environments. Using this approach we created four simulations: in the first one we use only grid brokers by the Meta-Broker (denoted by MB in the figures) to reach grid resources of the simulated Grids. In the second, third and fourth case we extend the matchmaking of the Meta-Broker (in order to simulate the whole SSV): when the background load of the selected grid broker exceeds 113%, it selects the Cloud broker instead. In the second case the Cloud broker has four pre-deployed VMs (4VMs), while in the third case only one, and later creates three more as described before (1+3VMs), and in the fourth it has one pre-deployed and creates 7 more on demand (1+3+4VMs). In Figure 6 and 7 we can see the evaluation results denoting the average and detailed execution times of the service requests respectively.

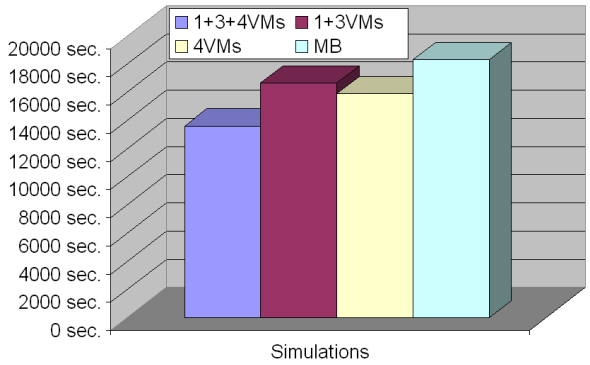


Figure 6. Average request run times.

From these results we can clearly see that the simulated SSV architecture overperforms the former (purely) Grid meta-brokering solution. Comparing the different deployment strategies we can see that on demand deployment introduces some overhead (4VMs was faster then 1+3VMs), but service duplication (1+3+4VMs) can enhance the performance and help to avoid SLA violations with additional VM deployment costs.

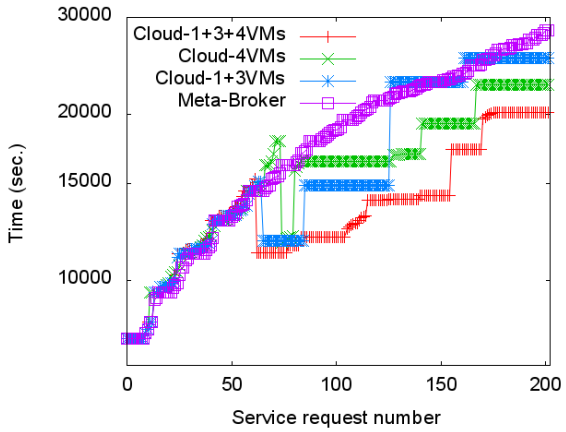


Figure 7. Detailed run times of requests.

V. CONCLUSION

In heterogeneous, distributed service-based environments such as Grids and Clouds, there is an emerging need for transparent, business-oriented autonomic service execution. In the future more and more companies will face the problem of unforeseen, occasional demand for a high number of computing resources. In this paper we have investigated how such problems could arise, and proposed a novel approach called Service-level agreement-based Service Virtualization (SSV). The presented general, conceptual architecture is built on three main components: the Meta-Negotiator responsible for agreement negotiations, the Meta-Broker for selecting the proper execution environment, and the Automatic Service Deployer for service virtualization and on-demand deployment. We have also discussed how the principles of autonomic computing are incorporated to the SSV architecture to cope with the error-prone virtualization environments. The proposed service virtualization architecture is validated in a simulation environment based on CloudSim, using a general biochemical application as a case study. The evaluation results clearly fulfill the expected utilization gains compared to a less heterogeneous Grid solution.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube), and from EGI-InSPIRE project (contract number RI-261323) and the Vienna Science and Technology Fund

(WWTF) under agreement ICT08-018, FoSII – Foundations of Self-governing ICT Infrastructures.

REFERENCES

[1] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Acs, A. Kertesz, G. Kecskemeti, S. Dustdar. LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures. In *Proc. of the First IEEE International Workshop on Emerging Applications for Cloud Computing*, Seoul, Korea, 2010.

[2] I. Brandic, D. Music, S. Dustdar. Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services. In *Proceedings of Grids meet Autonomic Computing Workshop*. ACM. 2009.

[3] I. Brandic, D. Music, P. Leitner, S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. In *the 6th International Workshop on Grid Economics and Business Models 2009 (Gecon09)*, 2009.

[4] R. Buyya, R. Ranjan and R. N. Calheiros, Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities, in proc. of the 7th High Performance Computing and Simulation Conference, 2009.

[5] Enabling Grids for E-sciencE (EGEE) project website: http://public.eu-egee.org/

[6] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 2009.

[7] R. Howard and L. Kerschberg. A knowledge-based framework for dynamic semantic web services brokering and management. In *DEXA '04: Proceedings of the Database and Expert Systems Applications, 15th International Workshop*, pp. 174–178, IEEE Computer Society, 2004.

[8] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, 2005.

[9] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. Automatic service deployment using virtualisation. In *Proc. of 16th Euromicro International Conference on Parallel, Distributed and network-based Processing*. IEEE Computer Society. 2008.

[10] J.O. Kephart, D.M. Chess. The vision of autonomic computing. *Computer* . 36:(1) pp. 41-50, Jan 2003.

[11] A. Kertesz, G. Kecskemeti, and I. Brandic. An SLA-based resource virtualization approach for on-demand service provision. In *Proceedings of the 3rd international Workshop on Virtualization Technologies in Distributed Computing (Barcelona, Spain, June 15 - 15, 2009)*. ACM, New York, pp. 27-34, 2009.

[12] A. Kertesz, G. Kecskemeti, and I. Brandic. Autonomic Resource Virtualization in Cloud-like Environments. In *Technical Report, TUV-1841-2009-04*. Distributed Systems Group, Institute for Information Systems, Vienna University of Technology, 2009.

[13] A. Kertesz and P. Kacsuk. GMBS: A New Middleware Service for Making Grids Interoperable. *Future Generation Computer Systems*, Volume 26, Issue 4, pp. 542–553, 2010.

[14] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2004. IEEE Computer Society.

[15] Z. Li and M. Parashar. An infrastructure for dynamic composition of grid services. In *GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 315–316, Washington, DC, USA, 2006. IEEE Computer Society.

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *CCGRID*, pages 124–131. IEEE Computer Society, 2009.

[17] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Proceedings of the 2005 European Grid Computing Conference (EGC 2005)*, February 2005.

[18] M. Parkin, D. Kuo, J. Brooke, and A. MacCulloch. Challenges in eu grid contracts. In *Proceedings of the 4th eChallenges Conference*, pp. 67–75, 2006.

[19] D. M. Quan and J. Altmann. Mapping a group of jobs in the error recovery of the grid-based workflow within sla context. *Advanced Information Networking and Applications, International Conference on*, 0:986–993, 2007.

[20] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *In Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.

[21] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with gria – industrial applications on a web services grid. In *E-SCIENCE '05: Proc. of the First International Conference on e-Science and Grid Computing*, pp. 98–105, 2005. IEEE Computer Society.

[22] H. N. Van, F. D. Tran, and J. Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8, 2009.

[23] C. A. Yfoulis, and A. Gounaris. Honoring SLAs on cloud computing services: a control perspective. In *Proceedings of the European Control Conference*, 2009.

[24] TINKER Conformer Generator workflow: http://www.lpds.sztaki.hu/gasuc/index.php?m=6&r=12

[25] The Grid Workloads Archive website: http://gwa.ewi.tudelft.nl

[26] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, 2006.