Appendices

A An SLA-based resource virtualization approach for on-demand service provision

An SLA-based Resource Virtualization Approach For On-demand Service Provision

Attila Kertesz
MTA SZTAKI
1518 Budapest, Hungary
P.O. Box 63.
attila.kertesz@sztaki.hu

Gabor Kecskemeti
MTA SZTAKI
1518 Budapest, Hungary
P.O. Box 63.
kecskemeti@sztaki.hu

Ivona Brandic
TU Vienna
1040 Vienna, Austria
Argentinierstr. 8/181-1
ivona@infosys.tuwien.ac.at

ABSTRACT

Cloud computing is a newly emerged research infrastructure that builds on the latest achievements of diverse research areas, such as Grid computing, Service-oriented computing, business processes and virtualization. In this paper we present an architecture for SLA-based resource virtualization that provides an extensive solution for executing user applications in Clouds. This work represents 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 architecture description focuses on three topics: agreement negotiation, service brokering and deployment using virtualization. The contribution is also demonstrated with a real-world case study.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks, Distributed Systems

General Terms

Design, Languages, Management, Experimentation

Keywords

SLA negotiation, Resource virtualization, Service Brokering, On-demand deployment

1. INTRODUCTION

Grid Computing [16] has succeeded in establishing production Grids serving various user communities all around the world. The emerging Web technologies have already affected Grid development; the latest solutions from related research fields (e.g. autonomous computing, P2P, etc.) also need to be considered in order to successfully transform the currently separated production Grids and Service oriented Architectures to the Internet of Services [25]. Cloud Computing [5] is a novel candidate that aims at creating this synergy; therefore we consider a cloud-like architecture focusing on agreement negotiation, service brokering and deployment using advanced virtualization techniques. Both Grids and Service Based Applications (SBAs) already provide solutions for executing complex user tasks. The web service model is based on three actors: a service provider, a service requester and a service broker [29]. Solutions building on this model use well-established and widely used technologies [29] that enable the collaboration of these three parties to fulfill service executions required by users. The newly emerging demands of users and researchers call for expanding this service model with business-oriented utilization (agreement handling) and support for human-provided and computation-intensive Grid services. Most of related works consider either virtualization approaches [12] [24] [18] without taking care of Service-Level Agreements (SLAs) or concentrates on SLA management neglecting the appropriate resource virtualizations [27] [6].

In this paper we propose a novel holistic architecture considering resource provision using the virtualization approach and combining it with the business-oriented utilization used for the SLA agreement handling. Thus, we provide an integrative infrastructure for on demand service provision based on SLAs. The main contributions of this paper include: (i) presentation of the novel architecture for the SLA-based resource virtualization and on-demand service provision, (ii) description of the architecture including *meta-negotiation, meta-brokering, brokering and automatic service deployment* and (iii) *demonstration* of the presented approach based on a case study. In the following section we summarize related works, in Section 3 we introduce the overall architecture and define the participating components and list the general utilization steps. In Section 4 the components of the three problem areas are detailed, and Section 5 presents a case study using the architecture. Finally Section 6 concludes the paper.

2. RELATED WORK

Though cloud-based service execution is rarely studied yet, some related works have already started to investigate, how business needs and more dynamicity could be represented in the web service model. The envisioned framework in [10] proposes a solution to extend this 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

VTDC'09, June 15, 2009, Barcelona, Spain.

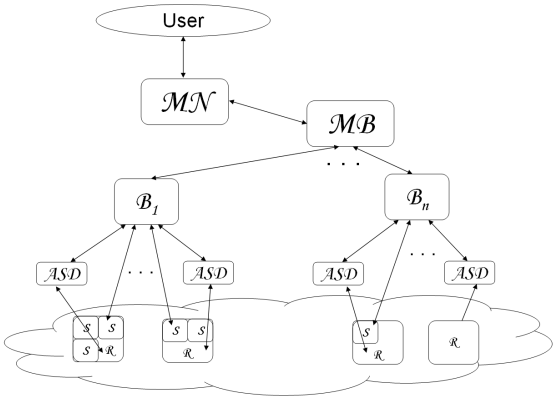


Figure 1: SRV architecture.

approaches. Works presented in [23] [21] discusses incorporation of SLA-based resource brokering into existing Grid systems. The Rudder framework [19] 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 [31], however, it is assumed that both partners understand the alternate offers protocol.

Regarding meta-brokering, LA Grid [26] developers aim at supporting grid applications with resources located and managed in different domains. They define broker instances, each of them collects resource information from its neighbors and save the information in its resource repository. The Koala grid scheduler [11] was redesigned to inter-connect different grid domains. They use a so-called delegated matchmaking (DMM), where Koala instances delegate resource information in a peer-2-peer manner. Gridway introduced a Scheduling Architectures Taxonomy [30], where Gridway instances can communicate and interact through grid gateways. These instances can access resources belonging to different Grid domains. Comparing the previous approaches, we can see that all of them use high level brokering that delegate resource information among different domains, broker instances or gateways. These solutions are almost exclusively used in Grids, they cannot co-operate with different brokers operating in pure service-based or cloud infrastructures. On the contrary, our proposed Meta-Broker can manage diverse, separated brokers.

Finally service deployment solutions are focusing on the deployment process itself and do not leverage their benefits on higher level. For example the Workspace Service (WS) [12] as a Globus incubator project supports wide range of scenarios involving virtual workspaces, virtual clusters and service deployment from installing a large service stack like ATLAS to deploy a single WSRF service if the Virtual Machine (VM) image of the service is available. The WS is designed to support several virtual machines – XEN [3], VMWare, VServer – to accomplish its task. Then the XenoServer open platform [24] is an open distributed architecture based on the XEN virtualization technique. It is aiming at global public computing. The platform provides services for server lookup, registry, distributed storage and a widely available virtualization server. Also the VMPlants [18] project proposes an automated virtual machine configuration and creation service which is heavily dependent on software dependency graphs. This project stays within cluster boundaries.

3. SLA-BASED RESOURCE VIRTUALIZATION (SRV) ARCHITECTURE

In this paper we present a unified service architecture that builds on three main areas: agreement negotiation, brokering and service deployment using virtualization. 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. First we introduce the general architecture naming the novelties and open issues, then we detail the aforementioned three main areas with respect to the shown architecture. Figure 1 shows our proposed, general architecture. Next we define the actors of the architecture:

- 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 fulfillment and violation.
- MB – Meta-Broker: Its role is to select a broker that is capable of deploying a service with the specified user requirements.
- 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.
- S – Service: The service that users want to deploy and/or execute.
- R – Resource: Physical machines, on which virtual machines can be deployed/installed.

The following detailed steps are done during utilization with respect to the steps shown in Figure 2:

- User starts a negotiation for executing a service with certain QoS requirements (specified in a Service Description (SD) with an SLA) (step 1)
- MN asks MB, if it could execute the service with the specified requirements (step 2)
- MB matches the requirements to the properties of the available brokers and replies with an acceptance or a different offer for renegotiation (step 3)
- MN replies with the answer of MB. Steps 1-4 may continue for renegotiations until both sides agree on the terms (to be written to an SLA document)
- User calls the service with the SD and SLA (step 5)
- MN passes SD and the possibly transformed SLA (to the protocol the selected broker understands) to the MB (step 6)

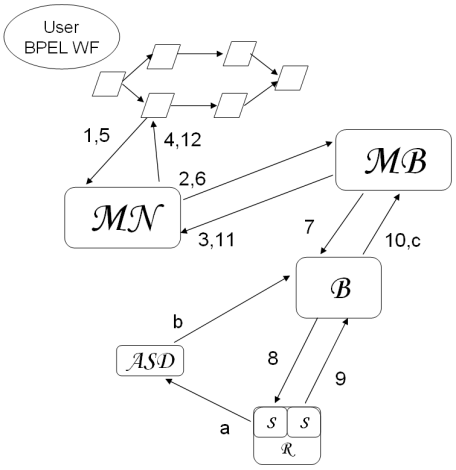


Figure 2: Detailed steps during SRV utilization.

- MB calls the selected Broker with SLA and a possibly translated SD (to the language of the Broker) (step 7)
- The Broker executes the service with respect to the terms of the SLA (step 8)
- In steps 9, 10, 11 and 12, the result of the execution is reported to the Broker, the MB, the MN, finally to the User (or workflow engine)
- ASD monitors the states of the virtual resources and deployed services (step a)
- ASD reports service availability and properties to its Broker (step b)
- All Brokers report available service properties to the MB (step c)

The previously presented sample architecture and the detailed utilization steps show that agreement negotiation, brokering and service deployment are closely related and each of them requires extended capabilities in order to interoperate smoothly. Nevertheless each part represents an open issue, since agreement negotiation, SLA-based service brokering and on-demand adaptive service deployment are not supported by current solutions in cloud-like environments.

4. REQUIREMENTS AND SOLUTIONS TO REALIZE SRV

In this section we detail three main categories, where the basic requirements of SRV-like systems arise. We place these areas in the SRV architecture shown in Figure 1, and detail the related parts of the proposed solution. We also emphasize the interactions among these components in order to build one coherent system.

In our proposed approach users describe the requirements for an SLA negotiation on a high level using the concept of meta-negotiations. During the meta-negotiation only those services are selected, which understand specific SLA document language and negotiation strategy or provide a specific security infrastructure. After the meta-negotiation process, a meta-broker selects a broker that is capable of deploying a service with the specified user requirements. Thereafter, the selected broker negotiates with virtual or physical resources using the requested SLA document language and using the specified negotiation strategy. Once the SLA negotiation is concluded, service can be deployed on the selected resource using the virtualization approach.

4.1 Agreement negotiation

As shown in Figure 1 to realize such a system we need to provide the following means of negotiation:

- User – MN: the User supplies a general meta-negotiation document
- MN – MB: they agree on specific negotiation documents containing specific negotiation strategy to be used, negotiation protocols to be used (WSLA, WS-Ag,) , terms of negotiation (e.g. time, price,), security infrastructure to be used
- MB – B: they agree on a specific SLA written in a specific SLA language (e.g. WSLA, WS-Agreement) containing concrete SLA parameters like concrete execution time, concrete price, etc.
- B – ASD: they agree on a specific service to be available on the ASD managed resources with the resource constraints resulted from the higher level negotiation – the service is going to be able to use the requested resources without disruptions from other parties
- Furthermore we need on each level (MN, MB, B, ASD) a negotiator which is responsible for generating and interpreting SLAs.

Before committing themselves to an SLA, the user and the provider may enter into negotiations that determine the definition and measurement of user QoS parameters, and the rewards and penalties for meeting and violating them respectively. The term negotiation strategy represents the logic used by a partner to decide which provider or consumer satisfies his needs best. A negotiation protocol represents the exchange of messages during the negotiation process. Recently, many researchers have proposed different protocols and strategies for SLA negotiation in Grids [22]. However, these not only assume that the parties to the negotiation understand a common protocol but also assume that they share a common perception about the goods or services under negotiation. In reality however, a participant may prefer to negotiate using certain protocols for which it has developed better strategies, over others. Thus, the parties to a negotiation may not share the same understanding that is assumed by the earlier publications in this space.

In order to bridge the gap between different negotiation protocols and scenarios, we propose a so-called meta-negotiation architecture [4]. *Meta-negotiation* is needed by means of a meta-negotiation document where participating parties may express: the pre-requisites to be satisfied for a negotiation, for example a specific authentication method required or terms they want to negotiate on (e.g. time, price, reliability); the negotiation protocols and document languages for the specification of SLAs that they support; and conditions for the establishment of an agreement, for example, a

```
1. <meta-negotiation ...>
2.   ...
3.  <pre-requisite>
4.   <security>
5.    <authentication value="GSI" location="uri"/>
6.   </security>
7.   <negotiation-terms>
8.    <negotiation-term name="beginTime"/>
9.    <negotiation-term name="endTime"/>
10.    ...
11.  </negotiation-terms>
12. </pre-requisite>
13. <negotiation>
14.  <document name="WSLA" value="uri" .../>
15.  <protocol name="alternateOffers"
16.    schema="uri" location="uri" .../>
17. </negotiation>
18. <agreement>
19.  <confirmation name="arbitrationService" value="uri"/>
20. </agreement>
21.</meta-negotiation>
```

Figure 3: Example Meta Negotiation Document

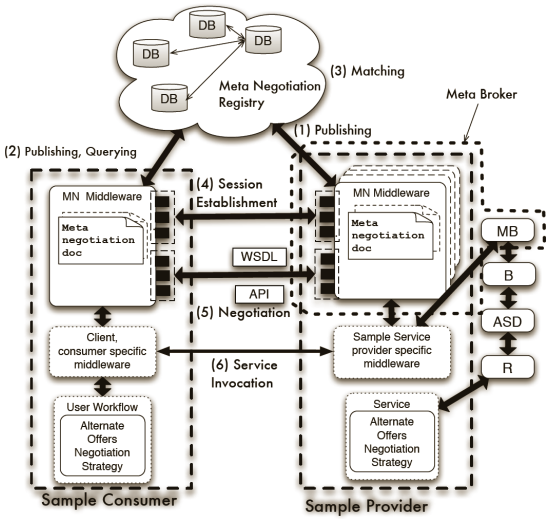


Figure 4: Meta-negotiation in SRV.

required third-party arbitrator. These documents are published into a searchable registry through which participants can discover suitable partners for conducting negotiations. In our approach, the participating parties publish only the protocols and terms while keeping negotiation strategies hidden from potential partners.

The participants publishing into the registry follow a common document structure that makes it easy to discover matching documents (as shown in Figure 3). This document structure consists of the following main sections: Each document is enclosed within the <meta-negotiation> ... </meta-negotiation> tags. The document contains an <entity> element defining contact information, organization and a unique ID of the participant. Each meta-negotiation comprises three distinguishing parts, namely *pre-requisites, negotiation and agreement* as described in the following paragraph.

As shown in Figure 3 prerequisites define the role a participating party takes in a negotiation, the security credentials and the negotiation terms. For example, the security element specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation process. For example, the consumer requires that the other party should be authenticated through the Grid Security Infrastructure (GSI) [7]. The negotiation terms specify QoS attributes that a party is willing to negotiate and are specified in the <negotiation-term> element. As an example, the negotiation terms of the consumer are beginTime, endTime, and price. Details about the negotiation process are defined within the <negotiation> element. Each document language is specified within <document> element. Once the negotiation has concluded and if both parties agree to the terms, then they have to sign an agreement. This agreement may be verified by a third party organization or may be lodged with another institution who will also arbitrate in case of a dispute. Figure 4 emphasizes a meta-negotiation infrastructure embedded into the agreement negotiation, brokering and service deployment architecture as proposed in Figure 1. In the following we explain the Meta-Negotiation infrastructure.

The registry is a searchable repository for meta-negotiation documents that are created by the participants. The meta-negotiation middleware facilitates the publishing of the meta-negotiation documents into the registry and the integration of the meta-negotiation framework into the existing client and/or service infrastructure, including, for example, negotiation or security clients. Besides being as a client for publishing and querying meta-negotiation documents (steps 1 and 2 in Figure 4), the middleware delivers necessary information for the existing negotiation clients, i.e. information for the establishment of the negotiation sessions (step 4, Figure 4) and information necessary to start a negotiation (step 5 in Figure 4).

4.2 Service brokering

In this subsection we are focusing on brokering-related aspects of the SRV architecture introduced in Section 2. Brokers (B) are the basic components that are responsible for finding the required services with the help of ASD. This task requires various activities, such as service discovery, matchmaking and interactions with information systems, service registries, repositories. There are several brokering solutions both in Grid [17] and SOAs [20], but agreement support is still an open issue. In our architecture brokers need to interact with ASDs and use adaptive mechanisms in order to fulfill the agreement (further requirements and interoperation is detailed in Section 4.3).

A higher-level component is also responsible for brokering in our architecture: the Meta-Broker (MB) [14]. *Meta-brokering* means a higher level resource management that utilizes existing resource or service brokers to access various resources. In a more generalized way, it acts as a mediator between users or higher level tools (e.g. negotiators or workflow managers) and environment-specific resource managers. The main tasks of this component are: to *gather* static and dynamic broker properties (availability, performance, provided and deployable services, resources, and dynamic QoS properties related to service execution), to *interact* with MN to create agreements for service calls, and to *schedule* these service calls to lower level brokers, i.e. match service descriptions (SD) to broker properties (which includes broker provided services). Finally the service call needs to be *forwarded* to the selected broker.

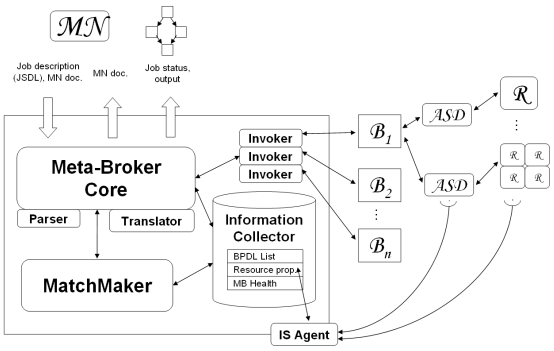


Figure 5: Meta-Broker in SRV.

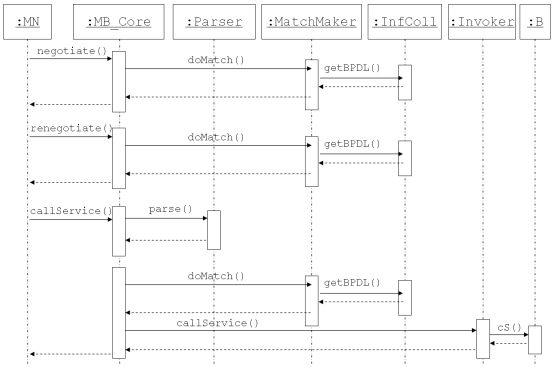


Figure 6: Interactions of the components of the Meta-Broker during utilization.

Figure 5 details the Meta-Broker (MB) architecture showing the required components to fulfill the above mentioned tasks. Different brokers use different service or resource specification descriptions for understanding the user request. These documents need to be written by the users to specify all kinds of service-related requirements. In case of resource utilization in Grids, OGF [1] has developed a resource specification language standard called JSDL [2]. As the JSDL is general enough to describe jobs and services of different grids and brokers, this is the default description format of MB. The *Translator* component of the Meta-Broker is responsible for translating the resource specification defined by the user to the language of the appropriate resource broker that MB selects to use for a given call. These brokers have various features for supporting different user needs, therefore an extendable Broker Property Description Language (BPDL) [15] is needed to express metadata about brokers and their offered services. The *Information Collector* (IC) component of MB stores the data of the reachable brokers and historical data of the previous submissions. This information shows whether the chosen broker is available, or how reliable its services are. During broker utilization the successful submissions and failures are tracked, and regarding these events a rank is modified for each special attribute in the BPDL of the appropriate broker (these attributes were listed above). In this way, the BPDL documents represent and store the dynamic states of the brokers. In order to support load balancing, there is an *IS Agent* (IS refers to Information System) reporting to IC, which regularly checks the load of the underlying resources of each connected broker, and store this data. It also communicates with the ASDs, and receives up-to-date data about the available services and predicted invocation times (that are used in the negotiations). The matchmaking process consists of the following steps: The *MatchMaker* (MM) compares the received descriptions to the BPDL of the registered brokers. This selection determines a group of brokers that can provide the required service. Otherwise the request is rejected. In the second phase the MM counts a rank for each of the remaining brokers. This rank is calculated from the broker properties that the IS Agent updates regularly, and from the service completion rate that is updated in the BPDL for each broker. When all the ranks are counted, the list of the brokers is ordered by these ranks. Finally the first broker of the priority list is selected, and the *Invoker* component forwards the call to the broker.

As previously mentioned, three main tasks need to be done by MB. The first, namely the information gathering, is done by the IS Agent, the second one is negotiation handling and the third one is service selection (illustrated in Figure 6). They need the following steps: During the negotiation process the MB interacts with MN: it receives a service request with the service description (in JSDL) and SLA terms (in MN document) and looks for a deployed service reachable by some broker that is able to fulfill 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.

4.3 Service deployment and virtualization

Automatic service deployment (ASD) is a higher-level service management concept, which provides the dynamics to SBAs – e.g. during the SBA's lifecycle services can appear and disappear without the disruption of their overall behavior.

Figure 7 shows the ASD's related components to the SRV architecture and their connections. To interface with a broker the ASD should be built on a repository (as an example it can use the Application Content Service – ACS [9] – standard proposed by the OGF [1]). All the master copies of all the deployable services should be stored in the repository. In this context the master copy means everything what is needed in order to deploy a service on a selected site – which we call the virtual appliance (VA). The virtual appliance could be either defined by an external entity or the ASD solution should be capable of acquiring it from an already running system. The repository allows the broker to determine which services are available for deployment and which are the static ones. Thus the repository would help to define a schedule to execute a service request taking into consideration those sites where the service has been deployed and where it could be executed but has not yet been installed (it is also monitored by the IS Agent of the Meta-Broker detailed in Section 4.2). If the deployed services are not available, it checks whether any of the latter resources can deliver the service taking into account the deployment cost.

The *Workspace Service* (WS), offers the virtualization capabilities – virtual machine creation, removal and manage-

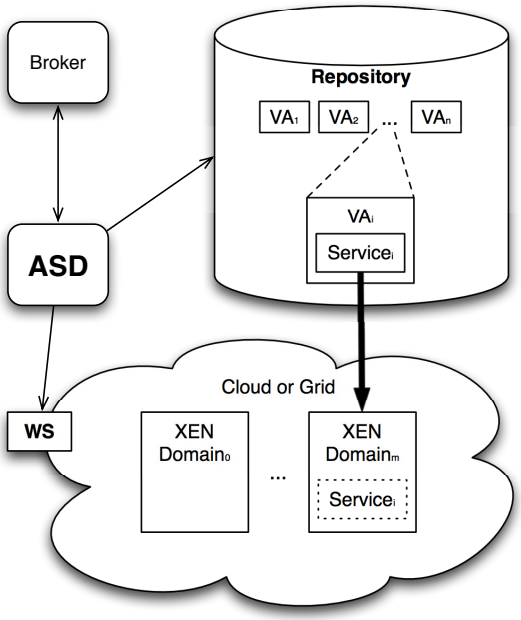


Figure 7: Service deployment in SRV.

ment – of a given site as a WSRF service. According to the OGSA-Execution Planning Services (EPS) [8] scenarios, a typical service broker has two main connections with the outside world: the *Candidate Set Generators* (CSG), and the Information Services. The task of the CSG is to offer a list of sites, which can perform the requested service according to the SLA and other requirements. Meanwhile the information services should offer general overview about the state of the SBA, Grid or Cloud. In most of the cases the candidate set generator is an integral part of the broker thus instead of the candidate set adjustments, the *broker* queries the candidate site list as it would do without ASD. Then the broker would evaluate the list and as a possible result to the evaluation it would initiate the deployment of a given service. As a result the service call will be executed as a composed service instead of a regular call. The composition will contain the deployment task as its starting point and the actual service call as its dependent task. Since both the CSG and the brokers heavily rely on the *information system*, the ASD can influence their decision through publishing dynamic data. This data could state service presence on sites where the service is not even deployed.

The selected *placement* of the ASD depends on the site policies on which the brokering takes place. In case the site policy requires a strict scheduling solution then either the CSG or the information system can be our target. If there is no restriction then the current broker can be replaced with an ASD extended one. In case the *candidate set generator* is altered then it should be a distributed, ontology-based adaptive classifier to define a set of resources on which the service call can be executed [28]. The CSG can build its classification rules using the specific attributes of the local information systems. Each CSG could have a feedback about the performance of the schedule made upon its candidates. The ASD extended CSG should have three interfaces to interoperate with other CSGs and the broker. First of all, the CSGs could form a P2P network, which requires two interfaces. The first manages the ontology of the different information systems by sharing the classifier rules and the common ontology patterns distributed as an OWL schema. The second interface supports decision-making among the peers. It enables the forwarding of the candidate request from the broker. The third interface lies between the broker and the CSGs to support passing the feedback for the supervised learning technique applied by the CSGs. This interface makes it possible for the broker to send back a metric packet about the success rate of the candidates.

The *brokers* should be adapted to ASD differently depending on where the ASD extensions are placed. If both the CSG's and the broker's behavior is changed then the broker can make smarter decisions. After receiving the candidate site set, the broker estimates the deployment and usage costs of the given service per candidate. For the estimation it queries the workspace service (WS). This service should accept cost estimate queries with a repository entry (VA) reference as an input. The ASD should support different agents discovering different aspects of the deployment. If only the broker's behavior is changed, and the CSG remains untouched, then the ASD would generate deployment service calls on overloaded situations (e.g. when SLA requirements are endangered). These deployment service calls should use the workspace service with the overloaded service's repository reference.

Finally it is possible to alter the *information system's behavior*. This way the ASD provides information sources of sites, which can accept service calls after deployment. The ASD estimates and publishes the performance related entries (like estimated response time) of the information system. These entries are estimated for each service and site pair, and only those are published which are over a predefined threshold.

Regarding component interactions, the ASD needs to be extended with the following in order to communicate with brokers: Requested service constraints have to be forced independently from what Virtual Machine Monitor (or hypervisor [3]) is used. To help the brokers making their decisions about which site should be used the ASD has to offer deployment cost metrics which can even be incorporated on higher level SLAs. The ASD might initiate service deployment/decommission on its own when it can prevent service usage peaks/lows, to do so it should be aware of the agreements made on higher levels.

5. CASE STUDY

In this section we discuss a case study on the *Maxillo Facial Surgery Simulation* (MFSS) in order to demonstrate the utilization of the presented architecture. The MFSS application facilitates the work of medical practitioners and provides the pre-operative virtual planning of maxillo-facial surgery. The application consists of a set of components running on local and different remote machines. These components may be organized as a workflow in order to simplify the work of the end users [4]. The main steps of the simulation are: (i) mesh generation is used for the generation of meshes necessary for the finite element simulation; (ii) mesh manipulation defines the initial and boundary conditions for the simulation; (iii) finite element analysis is a fully parallel MPI application usually running on a remote HPC cluster. In the followings we describe step by step how MFSS can

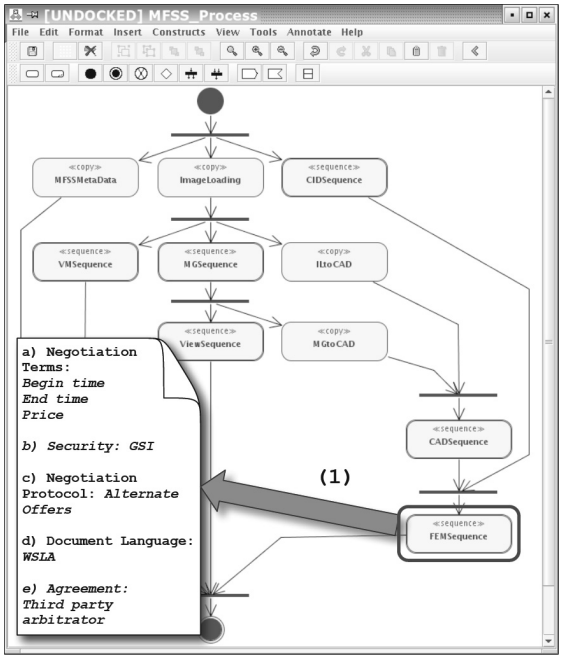


Figure 8: MFSS workflow with meta-negotiation specification.

be executed on the proposed architecture for the SLA-based Resource Virtualization. MFSS application can be modeled and executed using the Amadeus workflow tool, a QoS-aware Grid modeling, planning, and execution tool [4], see Figure 8.

As depicted in Figure 8, meta-negotiation for the MGSequence activity (used for mesh generation) is specified by means of (a) negotiation terms, (b) security restrictions, (c) negotiation protocols, (d) document languages and (e) preconditions for the agreement establishment. Negotiation terms are specified as begin time, end time, and price. In order to initiate a negotiation, GSI [7] security is required. The negotiation is performed based on the alternate offers protocol [32]. Therefore, the workflow application understands only the alternate offers protocol, and negotiation with resources which do not provide alternate offers protocol cannot be properly accomplished. Additional limitation considers document language used for the specification of SLAs. As shown in Figure 8, QoS is specified using WSLA [13]. The constraints shown in Figure 8 are transformed into a XML based meta-negotiation document. Thereafter this document is passed to the Meta-Broker. During the execution of the workflow, the Meta-Broker receives the service description in JSDL and the SLA terms in the meta-negotiation document. First a matchmaking process is started to select a broker that is able to execute the job with the specified requirements (resource requirements and agreement terms). The broker with the best performance values is selected, and the description and agreement is translated to the format understandable by the broker. Thereafter the broker is invoked with the transformed descriptions. The selected broker receives the descriptions and calls the ASD to deploy a service on a Cloud or a Grid, taking into account the cost requirements of the agreement, or chooses an already deployed, idle computing service. The job is executed and the results are returned to the workflow enactor. Finally the ASD decommissions the service.

6. CONCLUSIONS

In this paper a novel architecture for SLA-based resource virtualization with on-demand service deployment is introduced. The solution incorporates three enhancements: a meta-negotiation component for generic SLA management, a meta-brokering component for diverse broker management and an automatic service deployment for resource virtualization on the Cloud. We have stated the essential requirements for building the target architecture and demonstrated the utilization through a future case study. Our future work aims at finalizing the presented components and interfacing the architecture to commercial Clouds and production Grids.

7. 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 by the Vienna Science and Technology Fund (WWTF) under agreement ICT08-018, FoSII – Foundations of Self-governing ICT Infrastructures.

8. REFERENCES

[1] Open grid forum website. http://www.ogf.org, 1999.

[2] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job submission description language (jsdl) specification, version 1.0. Technical report, 2005. http://www.gridforum.org/documents/GFD.56.pdf.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[4] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya. Advanced qos methods for grid workflows based on meta-negotiations and sla-mappings. In *The 3rd Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, November 2008.

[5] 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.

[6] M. Q. Dang and J. Altmann. Resource allocation algorithm for light communication grid-based workflows within an sla context. *Int. J. Parallel Emerg. Distrib. Syst.*, 24(1):31–48, 2009.

[7] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, New York, NY, USA, 1998. ACM.

[8] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel,

F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Reich. The open grid services architecture, version 1.5. Technical report, 2006. http://www.ogf.org/documents/GFD.80.pdf.

[9] K. Fukui. Application contents service specification 1.0. Technical report, 2006. http://www.ogf.org/documents/GFD.73.pdf.

[10] 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*, pages 174–178, Washington, DC, USA, 2004. IEEE Computer Society.

[11] A. Iosup, T. Tannenbaum, M. Farrellee, D. Epema, and M. Livny. Inter-operating grids through delegated matchmaking. *Sci. Program.*, 16(2-3):233–253, 2008.

[12] 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.

[13] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, V11(1):57–81, March 2003.

[14] A. Kertesz and P. Kacsuk. Meta-broker for future generation grids: A new approach for a high-level interoperable resource management. In *Grid Middleware and Services Challenges and Solutions*, pages 53–63. Springer US, 2008.

[15] A. Kertesz, I. Rodero, and F. Guim. Data model for describing grid resource broker capabilities. In *Grid Middleware and Services Challenges and Solutions*, pages 39–52. Springer US, 2008.

[16] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[17] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[18] 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.

[19] 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.

[20] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004.

[21] 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.

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

[23] 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.

[24] 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.

[25] N. G. G. Report. Future for european grids: Grids and service oriented knowledge utilities – vision and research directions 2010 and beyond. Technical report, December 2006. ftp://ftp.cordis.lu/pub/ist/docs/grids/ngg3_eg_final.pdf.

[26] I. Rodero, F. Guim, J. Corbalan, L. Fong, Y. Liu, and S. Sadjadi. Looking for an evolution of grid scheduling: Meta-brokering. In *Grid Middleware and Services Challenges and Solutions*, pages 105–119. Springer US, 2008.

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

[28] M. Taylor, C. Matuszek, B. Klimt, and M. Witbrock. Autonomous classification of knowledge into an ontology. In *The 20th International FLAIRS Conference (FLAIRS)*, 2007.

[29] A. Tsalgatidou and T. Pilioura. An overview of standards and related technology in web services. *Distrib. Parallel Databases*, 12(2-3):135–162, 2002.

[30] T. Vazquez, E. Huedo, R. S. Montero, and I. M. Llorente. Evaluation of a utility computing model based on the federation of grid infrastructures. In *Euro-Par 2007 Parallel Processing*, pages 372–381. Springer Berlin / Heidelberg, 2007.

[31] 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.

[32] S. Venugopal, X. Chu, and R. Buyya. A negotiation mechanism for advance resource reservation using the alternate offers protocol. In *16th International Workshop on Quality of Service (IWQoS 2008)*, pages 40–49, June 2008.

B Adaptive scheduling solution for grid meta-brokering

Acta Cybernetica **00** (0000) 1–19.

Adaptive scheduling solution for grid meta-brokering*

Attila Kertész†, József Dániel Dombi‡ and József Dombi‡

Abstract

The nearly optimal, interoperable utilization of various grid resources play an important role in the world of grids. Though well-designed, evaluated and widely used resource brokers have been developed, these existing solutions still cannot cope with the high uncertainty ruling current grid systems. To ease the simultaneous utilization of different middleware systems, researchers need to revise current solutions. In this paper we propose advanced scheduling techniques with a weighted fitness function for an adaptive Meta-Brokering Grid Service, which enables a higher level utilization of the existing grid brokers. We also set up a grid simulation environment to demonstrate the efficiency of the proposed meta-level scheduling solution. The presented evaluation results show that the proposed novel scheduling technique in the meta-brokering context delivers better performance.

Keywords: Grid Computing, Meta-Brokering, Scheduling, Grid Service, Random Number Generator function

1 Introduction

Ten years ago a new computing infrastructure called the Grid was born. Ian Foster et. al. made this technology immortal by publishing the bible of the Grid [1] in 1998. Grid Computing has become an independent research field since then: currently grids are targeted by many world-wide projects. A decade is a long time: though the initial goal of grids to serve various scientific communities by providing a robust hardware and software environment is still unchanged, different middleware solutions have been developed (Globus Toolkit [2], EGEE [3], UNICORE [4], etc.). The realizations of these grid middleware systems formed production grids that are mature enough to serve scientists having computation and data intensive applications. Nowadays research directions are focusing on user needs: more efficient utilization and interoperability play the key roles. To solve these problems

*This work was supported by the FP7 Network of Excellence S-Cube funded by the European Commission (Contract FP7/2007-2013).

†MTA SZTAKI, University of Szeged E-mail: keratt@inf.u-szeged.hu

‡University of Szeged E-mail: {dombijd,dombi}@inf.u-szeged.hu

grid researchers have two options: as a member of a middleware developer group they can come up with new ideas or newly identified requirements and go through the long process of designing, standardizing and implementing the new feature, then waiting for the next release containing the solution. Researchers sitting on the other side or unwilling to wait for years for the new release, need to rely on the currently available interfaces of the middleware components and use advanced techniques of other related research fields (peer-to-peer, web computing, artificial intelligence, etc.). We have chosen the second option to improve grid resource utilization with an interoperable resource management service.

Since the management and advantageous utilization of highly dynamic grid resources cannot be handled by the users themselves, various grid resource management tools have been developed, supporting different grids. User requirements created certain properties that resource managers have learned to support. This development is still continuing, and users already need to stress themselves to distinguish brokers and to migrate their applications, when they move to a different grid. Interoperability problems and multi-broker utilization have emerged the need for higher level brokering solutions. The meta-brokering approach means a higher level resource management by enabling automatic and simultaneous utilization of grid brokers. Scheduling at this level requires sophisticated approaches, because high uncertainty presents at all stages of grid resource management. Despite these difficulties, this work addresses the resource management layer of middleware systems and proposes an enhanced scheduling technique to improve grid utilization in a high-level brokering service.

In the following sections of this paper we are focusing on a meta-brokering solution for grid resource management and present an adaptive scheduling technique that targets better scheduling in global grids. In Section 2 we introduce meta-brokering in grids, in Section 3 we describe our proposed scheduling solution, and in Section 4 we present our simulation architecture and the evaluation of our proposed solution. Finally, in Section 5 we gather the related research directions and Section 6 concludes the paper.

2 The need for grid meta-brokering

Heterogeneity appears not only in the fabric layer of grids, but also in the middleware. Even components and services of the same middleware may support different ways for accessing them. After a point this variety slows down grid development, and makes grid systems unmanageable. Most grid middleware systems have fixed interfaces to access their components and propagate information flow. In case of resource management most of the middleware systems provide access to static properties (number of CPUs, size of memory, etc.) and some also give dynamic ones (number of waiting jobs, expected response time), but this data is usually outdated due to timely periodic refreshing. As a result we can state that there is a high uncertainty in current grids, which is not likely to change soon. Though the first de facto middleware, the Globus Toolkit [2], did not have a resource broker that au-

tomates resource selection, the currently used middleware systems have built-in or supporting brokers [8]. The development of different brokers and grids has started a separation process in the research and user community, too. Therefore one of the major problems of current grids is grid interoperability. Focusing on the resource management layer of grids an obvious solution would be to interconnect brokers to create interoperability. Unfortunately current brokers do not have a common protocol and uniform interface for intercommunication, though the OGF-GSA [10] started to work on this issue. Once they standardize a solution we still would need to wait till all the brokers implement it in order to establish interoperability. In order to achieve this goal in a short term we have chosen to interconnect brokers by a high-level resource manager: we introduced meta-brokering (first proposed in [7]) that means a higher level utilization of the existing, widely used and reliable resource brokers. Since most of the users have certificates to access more Grids, they are facing the problem of grid selection: which grid, which broker should I choose for my specific application? Just like users needed resource brokers to choose proper resources within a grid, now they need a meta-brokering service to decide, which broker (or grid) is the best for them and also to hide the differences of utilizing them. In this way the meta-brokering approach solves the grid interoperability problem at the level of resource management by providing a uniform interface for the users of all the grids they have access to.

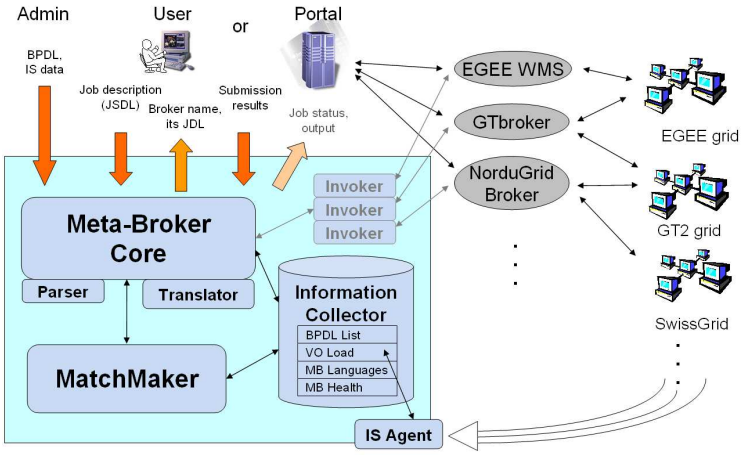


Figure 1: Components of the Grid Meta-Broker Service

Figure 1 introduces the revised architecture of the Grid Meta-Broker Service that enables the users to access resources of different grids through their own brokers. In this way, this higher level tool matches resource brokers to user requests. The system is implemented as a web-service that is independent from middleware-specific components. The provided services can be reached through WSDL (Web Services Description Language). In the following we give a brief summary of its com-

ponents and their operation. As the JSDL (Job Submission Description Language) standard [5] proposed by OGF (Open Grid Forum [9]) is general enough to describe jobs of different grids and brokers, we have chosen this to be the job description language of the Meta-Broker. The Translator components of the Meta-Broker are responsible for transforming the resource specification defined by the user to the language of the appropriate resource broker that the Meta-Broker selects to use for a given job. Regarding all the various job specification formats used by different grid middleware systems not all job attributes can be expressed in each document. Furthermore we revealed some useful scheduling-related attributes that are also missing from JSDL. To overcome these limitations we specified MBSDL (Meta-Broker Scheduling Description Language [6]). The main attribute categories are: middleware constraints, scheduling policies and QoS requirements. This schema can be used to extend JSDL with scheduling-related attributes. Besides describing user jobs, we also need to describe resource brokers in order to differentiate and manage them. These brokers have various features for supporting different user needs. These needs should be able to be expressed in the users JSDL, and identified by the Meta-Broker for each corresponding broker. Therefore we proposed an extendable BPDL (Broker Property Description Language [6]) – similar to the purpose of JSDL –, to express metadata about brokers. The common subset of the individual broker properties is stored here: the supported middleware, job types, certificates, job descriptions, interfaces, monitoring features and dynamic performance data. The scheduling-related ones are stored in MBSDL: fault tolerant features (checkpointing, rescheduling, replication), agreement support, scheduling policies (ranking by resource attributes) and QoS properties (e.g. advance reservation, co-allocation, email notification). The union of these properties forms a complete broker description document that can be filled out and regularly updated for each utilized resource broker. These two kinds of data formats are used by the Meta-Broker: JSDL is used by the users to specify jobs and the BPDL (Broker Property Description Language) by administrators to specify brokers – both parties can use MBSDL to extend their descriptions.

The Information Collector (IC) component of the Meta-Broker stores the data of the reachable brokers and historical data of the previous submissions. This information shows whether the chosen broker is available, or how reliable its services are. During broker utilization the successful submissions and failures are tracked, and regarding these events a rank is modified for each special attribute in the BPDL of the appropriate broker (these attributes were listed above). In this way, the BPDL documents represent and store the dynamic states of the brokers. All data is stored in XML, and advanced XML-serialization techniques are used by the IC. The load of the resources behind the brokers is also taken into account to help the Matchmaker to select the proper environment for the actual job. When too many similar jobs are needed to be handled by the Meta-Broker an eager matchmaking may flood a broker and its grid. That is the main reason why load balancing is an important issue. In order to cope with this problem, there is an IS (Information System) Agent component reporting to the Information Collector, which regularly checks the load of the underlying grids of each connected resource broker, and store this data. This

tool is implemented as separate web-service connected to the Information System of the grids behind the utilized brokers. With the additional information provided by this agent the matchmaking process can adapt to the load of the utilized grids. Finally, the actual state (load, configurations) of the Meta-Broker is also stored here, and it can also be queried by users. The continuous monitoring of grid load and broker performances makes this grid service self-adaptive.

The previously introduced languages are used for matching the user requests to the description of the interconnected brokers: which is the role of the Matchmaker component. The JSDL contains the user request (this supposed to be an exact specification of the user's job) using the extended attributes, while the interconnected brokers are described by their BPDL documents. The default matchmaking process consists of the following steps to find the fittest broker:

- The Matchmaker compares the JSDL of the actual job to the BPDL of the registered resource brokers. First the job requirement attributes are matched against the broker properties stored in their BPDLs: this selection determines a group of brokers that are able to submit the job. This phase consists of two steps: first the brokers are filtered by all the requirements stated in the JSDL. When none of the brokers can fulfill the request, another filtering process will be started with minimal requirements (those ones are kept which are real necessary for job execution). If the available brokers still can not accept the job, it will be rejected.
- In the second phase the previous submissions of the brokers and the load of the underlying grids are taken into account: The MatchMaker counts a rank for each of the remaining brokers. This rank is calculated from the load that the IS Agent regularly updates, and from the job completion rate that is updated in the PerformanceMetrics field of the BPDL for each broker. When all the ranks are counted, the list of the brokers is ordered by these ranks.
- Finally the first broker of the priority list is selected for submission.

3 Adaptive Scheduling for meta-brokering

In the previous section we introduced the Grid Meta-Broker and shown how the default matchmaking is carried out. The main goal of this paper is to enhance the scheduling part of this matchmaking process. To achieve this, we have created a Decision Maker component and inserted it into the MatchMaker component of the Meta-Broker (see Figure 1). The first part of the matchmaking is unchanged: the list of the available brokers is filtered according to the requirements of the actual job read from its JSDL. Then the list of the remaining brokers along with their performance data and background grid load are sent to the Decision Maker in order to determine the fittest broker for the actual job. The scheduling techniques and the process are described in the following paragraphs.

The Decision Maker uses a random number generator, and we chose a JAVA solution, which generates pseudorandom numbers. This generator produces exactly

the same sequence of random numbers for each execution with the same initial value. This initial value is called the seed. The JAVA random number generator class uses uniform distribution and 48-bit seed, which is modified by a linear congruential formula[11]. The default seed is the current time in milliseconds since 1970. We also developed a unique random number generator, which generates random numbers with a given distribution. We called this algorithm as generator function. In our case we defined a score value for each broker, and we created the distribution based on the score value. For example the broker which has the highest score number has the highest probability to be chosen. In this algorithm the inputs are the broker id and the broker score, which are integer numbers (see Table 1).

Table 1: Inputs of the algorithm

BrokerID	Score
3	2
4	3
5	1
6	2

The next step is to choose a broker and put it into a temporary array: the cardinality is determined by the score value (see Table 2). After the temporary array is filled, we shuffle this array and choose an array element using the JAVA random generator. In the example shown in Table 3 the generator function chose the broker with id 4.

Table 2: Elements in the temporary array

Broker ID	3	3	4	4	4	5	6	6
Array ID	1	2	3	4	5	6	7	8

Table 3: Shuffled temporary array

Broker ID	4	3	6	3	4	4	5	6
Array ID	1	2	3	4	5	6	7	8

Java Random generator: **5**

To improve the scheduling performance of the Meta-Broker we need to send the job to the broker that best fits the requirements and executes the job without failures with the shortest execution time. Every broker has three properties that

the algorithm can rely on: the successful counter, the failure counter and the load counter.

- The successful counter represents the number of jobs which had finished without any errors.
- The failure counter shows the number of failed jobs.
- The load counter indicates the actual load of the grid behind the broker (in percentage).

We have developed four different kinds of decision algorithms. The trivial algorithm uses only a random number generator to select a broker. The other three algorithms take into account the previously mentioned broker properties. These algorithms define a score number for each broker and use the generator function to select one. To calculate the score value we build a weighted sum of the evaluated properties. This number is always an integer number. Furthermore, the second and third decision algorithms take into account the maximum value of the failure and load counter. This means that we extract the maximum value of the properties before multiplying them with the weight. The generator function of the third algorithm chooses a broker which score number is not smaller than the half of the highest score value.

After testing different kinds of weighted systems, we conclude that the most useful weights are shown in Table 4) that represent the weights of the used decision algorithms.

Table 4: The weights of the decision makers

Decision Maker	Success_weight	Failed_weight	Load_weight
Decision I.	3	0.5	1
Decision II.	4	4	4
Decision III.	4	4	4

During the utilization of the Meta-Broker, the first two broker properties (successful and failure counter) are incremented through a feedback method that the simulator (or a user or portal in real world cases) calls after the job submission is finished. The third property, the load value, is queried by the Meta-Broker from an information provider (Information System) of a Grid. During simulation this data is saved to a database by the Broker entities of the simulator (described later and shown in Figure 2). This means by the time we start the evaluation and till we do not receive feedback from finished jobs the algorithms can only rely on the background load of the grids. To further enhance the scheduling we developed a training process that can be executed before the simulation in order to initialize the first and second properties. This process sends a small number of jobs with various properties to the brokers and set the successful and failed jobs number at

the BPDLs of the brokers. With this additional training method we expect shorter execution times by selecting more reliably brokers.

4 Evaluation

In order to evaluate our proposed scheduling solution, we have created a general simulation environment, in which all the related grid resource management entities can be simulated and coordinated. The GridSim toolkit [12] is a fully extendable, widely used and accepted grid simulation tool – these are the main reasons why we have chosen this toolkit for our simulations. It can be used for evaluating VO-based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global grids. It supports modeling and simulation of heterogeneous grid resources, users, applications, brokers and schedulers in a grid computing environment. It provides primitives for the creation of jobs (called gridlets), mapping these jobs to resources and their management, therefore resource schedulers can be simulated to study scheduling algorithms. GridSim provides a multilayered design architecture based on SimJava [13], a general purpose discrete-event simulation package implemented in Java. It is used for handling the interaction or events among GridSim components. All components in GridSim communicate with each other through message passing operations defined by SimJava.

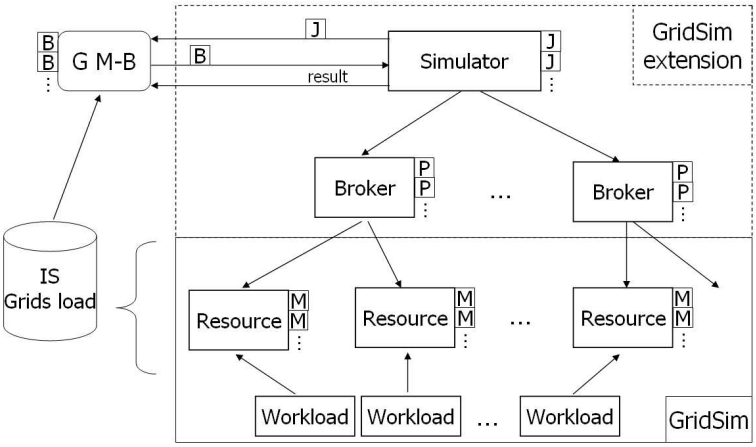


Figure 2: Meta-Brokering simulation environment based on GridSim

Our general simulation architecture can be seen in Figure 2. On the bottom-right part we can see the GridSim components used for the simulated grid systems. Resources can be defined with different grid-types. Resources consist of more machines, to which workloads can be set. On top of this simulated grid infrastructure we can set up brokers. The Broker and Simulator entities have been developed by us in order to enable the simulation of meta-brokering. Brokers are extended

GridUser entities:

- they can be connected to one or more resources;
- different properties can be set to these brokers (agreement handling, co-allocation, advance reservation, etc.);
- some properties can be marked as unreliable;
- various scheduling policies can be defined (pre-defined ones: rnd – random resource selection, fcpu – resources having more free cpus or less waiting jobs are selected, nfailed – resources having less machine failures are selected);
- generally resubmission is used, when a job fails due to resource failure;
- finally they report to the IS (Information System) Grid load database by calling the feedback method of the Meta-Broker with the results of the job submissions (this database has a similar purpose as a grid Information System).

The Simulator is an extended GridSim entity:

- it can generate a requested number of gridlets (jobs) with different properties, start and run time (length);
- it is connected to the created brokers and able to submit jobs to them;
- the default job distribution is the random broker selection (though at least the middleware types are taken into account);
- in case of job failures a different broker is selected for the actual job;
- it is also connected to the Grid Meta-Broker through its web service interface and able to call its matchmaking service for broker selection.

4.1 Preliminary testing phase

Table 5 shows the details of the preliminary evaluation environment. 10 brokers can be used in this simulation environment. The second column denotes the scheduling policies used by the brokers: fcpu means the jobs are scheduled to the resource with the most free cpus, nfail means those resources are selected that have less machine failures, and rnd means randomized resource selection. The third column shows the capabilities/properties (eg: coallocation, checkpointing, ...) of the brokers: three properties are used in this environment, subscript F means unreliability, a broker having such a property may fail to execute a job with the requested service with a probablity of 0.5. The fourth column contains the number of resources utilized by a broker, while the fifth column contains the number of background jobs submitted to the broker (SDSC BLUE workload logs taken from the Parallel Workloads Archive [14]) during the evaluation timeframe.

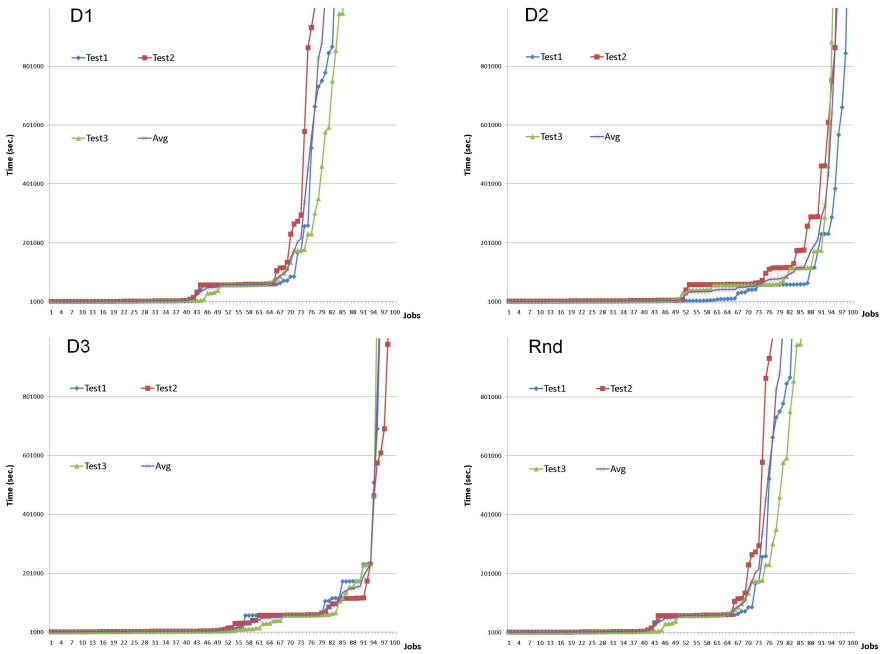


Figure 3: Diagrams of the preliminary evaluation for each algorithm

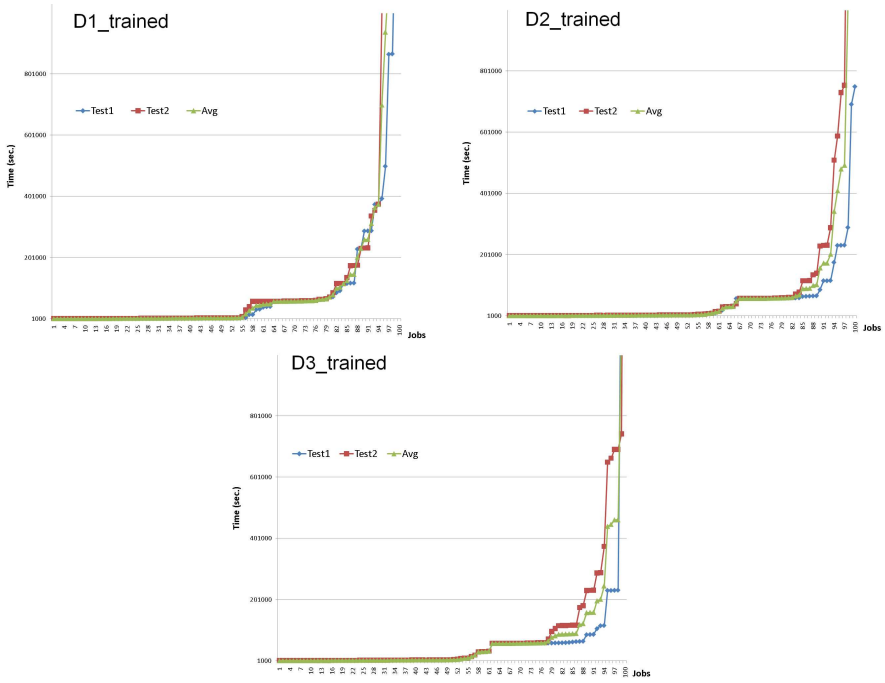


Figure 4: Diagrams of the preliminary evaluation for each algorithm with training phases

Table 5: Preliminary evaluation setup.

Broker	Scheduling	Properties	Resources	Workload
1.	fcpu	A	8	20*8
2.	fcpu	B	8	20*8
3.	fcpu	C	8	20*8
4.	fcpu	A_F	8	20*8
5.	fcpu	B_F	8	20*8
6.	fcpu	C_F	8	20*8
7.	nfail	$A_F B$	10	20*10
8.	nfail	AC_F	10	20*10
9.	nfail	$B_F C$	10	20*10
10.	rnd	-	16	20*16

As shown in the table we utilized 10 brokers to execute our first experiment. In this case we submitted 100 jobs to the system, and measured the makespan of all the jobs (time elapsed from submission till the successful finishing, including waiting time in the queue of the resources and resubmissions on failures). Out of the 100 jobs 40 had no special property (this means all the brokers can successfully execute them), for the rest of the jobs the three properties were distributed equally: 20 jobs had property A, 20 had B and 20 had C. Each resource of the simulated grids has been utilized by 20 background jobs (workload) with different submission times according to the distribution defined by the SDSC BLUE workload logs.

Figure 3 shows the detailed evaluation runs with the scheduling algorithms Decision 1 (D1), 2 (D2), 3 (D3) and without the use of the Meta-Broker (randomized broker selection – Rnd) respectively. Figure 4 shows the measured values with the three algorithms with training (we submitted 10 jobs to each broker to set their initial performance values). In Figure 5 we can see the averages of the tests with the different algorithms. This illustrates best the differences of the simulations with and without the use of the Meta-Broker.

After we have seen the diagrams of the preliminary evaluations we can state that all the proposed scheduling algorithms (D1, D2 and D3) provide shorter execution times than the random broker selection. In the main evaluation phases our goal was to set up a more realistic environment and to experience with a higher number of jobs.

4.2 Main testing phase

Table 6 shows the evaluation environment used in the main evaluation. The simulation setup was derived from real-life production grids: current grids and brokers support ony a few special properties: we used four. To determine the (proportional) number of resources in our simulated grids we compared the sizes of current production grids (EGEE VOs, DAS3, NGS, Grid5000, OSG, ...). We used the same

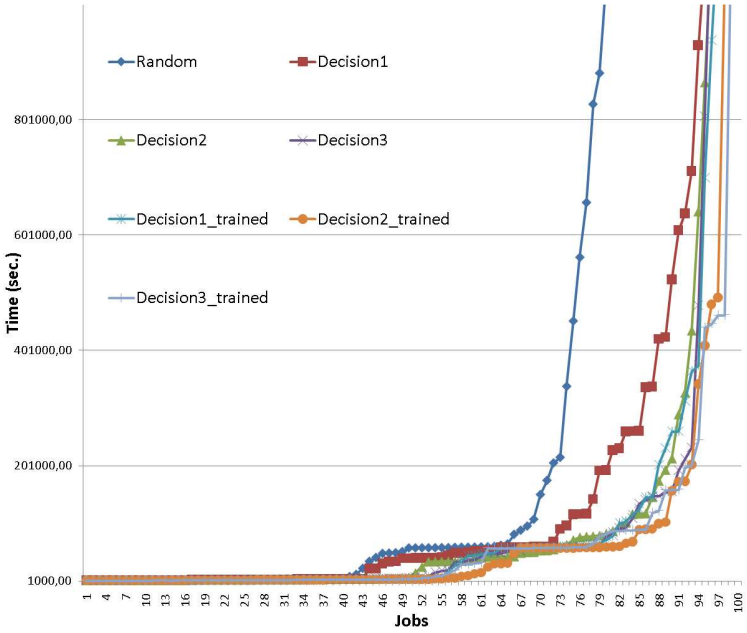


Figure 5: Summary diagram of the preliminary evaluation

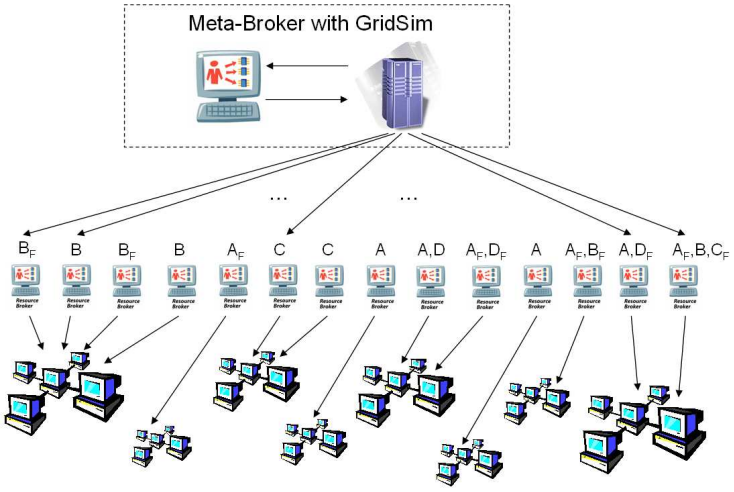


Figure 6: Simulation in the main evaluation environment

Table 6: Main evaluation setup.

Broker	Scheduling	Properties	Resources	Workload
1.	fcpu	A	6	50*6
2.	fcpu	A_F	8	50*8
3.	fcpu	A	12	50*12
4.	fcpu	B	10	50*10
5.	fcpu	B_F	10	50*10
6.	fcpu	B	12	50*12
7.	fcpu	B_F	12	50*12
8.	fcpu	C	4	50*4
9.	fcpu	C	4	50*4
10.	fcpu	$A_F D$	8	50*8
11.	fcpu	AD	10	50*10
12.	fcpu	AD_F	8	50*8
13.	fcpu	AB_F	6	50*6
14.	fcpu	ABC_F	10	50*10

notations in this table as before.

In the main evaluation we utilized 14 brokers. In this case we submitted 1000 jobs to the system, and again measured the makespan of all the jobs. Out of the 1000 jobs 100 had no special property, for the rest of the jobs the four properties were distributed in the following way: 30 jobs had property A, 30 had B, 20 had C and 10 had D. The workload logs contained 50 jobs for each resource. In the training processes 100 jobs were submitted to each broker prior the evaluations to set the initial values. Figure 6 shows the graphical representation of the simulation environment.

In the first phase of the main evaluation the simulator submitted all the jobs at once, just like in the preliminary evaluation. The results for this phase can be seen in Figure 7.

In the first phase we could not exploit all the features of the algorithms, because we submitted all the jobs at once and the performance data of the brokers were not updated early enough for the matchmaking process. To avoid this, in the last phase of the main evaluation we submitted the jobs periodically: 1/3 of the jobs were submitted in the beginning, then the simulator waited for 200 jobs to finish and update the performances of the brokers. After this the simulator submitted again 1/3 of all the jobs and waited for 300 more to finish. Finally the rest of the jobs (1/3 again) were submitted. In this way the broker performance results could be used by the scheduling algorithms. Figure 8 shows the results of the last evaluation phase. Here we can see that the runs with training could not make too much advantage of the trained values, because the feedback of the first submission period compensates the lack of training.

Figure 9 displays the summary of the different evaluation phases. The depicted

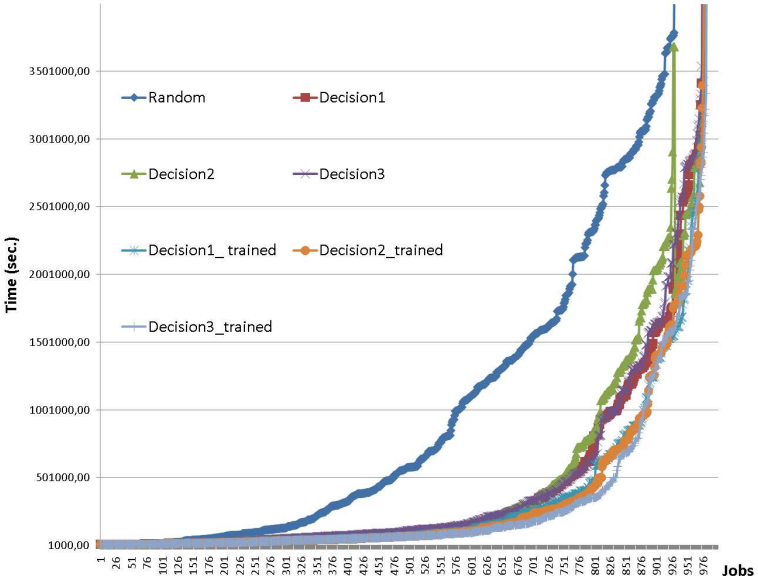


Figure 7: Diagram of the first phase of the main evaluation

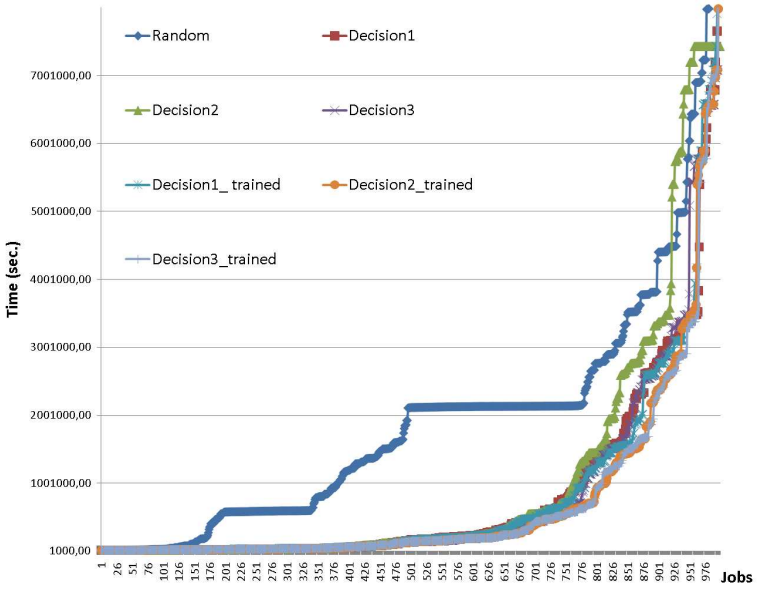


Figure 8: Diagram of the second phase of the main evaluation

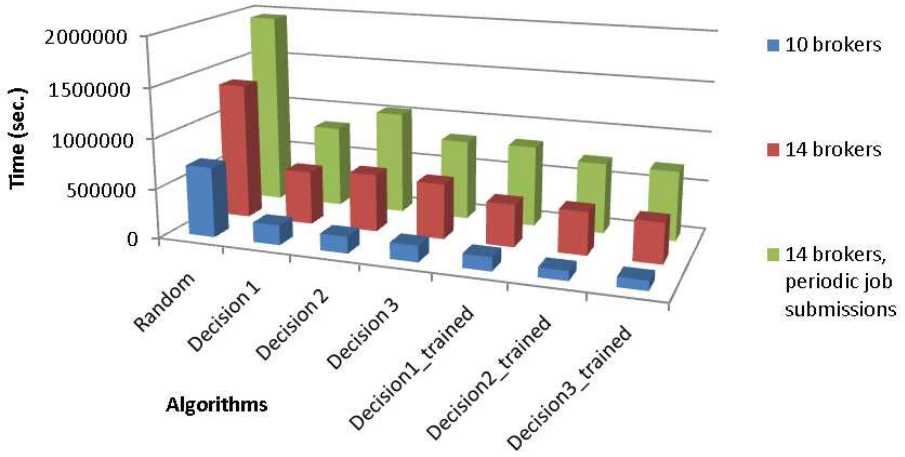


Figure 9: Summary of the evaluation results

columns show the average values of each evaluation runs with the same parameters. The results clearly show that the more intelligence (more sophisticated methods) we put into the system the higher performance we gain. The most advanced version of our proposed meta-brokering solution is the Decision Maker with the algorithm called Decision3 with training. Once the number of brokers and job properties will be high enough to set up this Grid Meta-Broker Service for inter-connecting several Grids, with the presented scheduling algorithms our service will be ready to serve thousands of users even under high uncertainty.

5 Related work

Meta-brokering means a higher level solution that brokers user requests among various grid domains. One of these promising approaches aims at enabling communication among existing resource brokers. The GSA-RG of OGF [10] is currently working on a project enabling grid scheduler interaction. They try to define common protocol and interface among schedulers enabling inter-grid usage. In this work they propose a Scheduling Description Language to extend the currently available job description language solutions. This work is still in progress, up to now only a partial SDL schema has been created. The meta-scheduling project in LA Grid [15] aims to support grid applications with resources located and managed in different domains. They define broker instances with a set of functional modules: connection management, resource management, job management and notification management. These modules implement the APIs and protocols used in LA Grid through web services. Each broker instance collects resource information from its neighbors and save the information in its resource repository or in-core memory. The resource information is distributed in the Grid and each instance will have a

view of all resources. The Koala grid scheduler [16] was designed to work on DAS-2 interacting with Globus [2] middleware services with the main features of data and processor co-allocation; lately it is being extended to support DAS-3 and Grid'5000. To inter-connect different grids, they have also decided to use inter-broker communication. Their policy is to use a remote grid only if the local one is saturated. In an ongoing experiment they use a so-called delegated matchmaking (DMM), where Koala instances delegate resource information in a peer-2-peer manner. Gridway introduces a Scheduling Architectures Taxonomy [17], where they describe a Multiple Grid Infrastructure. It consists of different categories, we are interested in the Multiple Meta-Scheduler Layers, where Gridway instances can communicate and interact through grid gateways. These instances can access resources belonging to different administrative domains (grids/VOs). They pass user requests to another domain when the current is overloaded – this approach follows the same idea as the previously introduced DMM. Gridway is also based on Globus, and they are experimenting with GT4 and gLite [3]. Comparing the previous approaches, we can see that all of them use a new method to expand current grid resource management boundaries. Meta-brokering appears in a sense that different domains are being examined as a whole, but they rather delegate resource information among domains, broker instances or gateways. Usually the local domain has preference, and when a job is passed to somewhere else, the result should be passed back to the initial point. Regarding multi-grid usage, the existing grids are very strict and conservative in the sense that they are very reluctant to introduce any modification that is coming from research or from other grid initiatives. Hence the solutions aiming at inter-connecting the existing brokers through common interfaces require a long standardization procedure before it will be accepted and adapted by the various grid communities. On the other hand the advantage of our proposed meta-brokering concept is that it does not require any modification of the existing grid schedulers, since it utilizes and delegates broker information by reaching them through their current interfaces. The HPC-Europa Project researchers also considered taking steps towards meta-brokering [18]; currently we have an ongoing work together with them to define a common meta-brokering model.

6 Conclusions

The Grid Meta-Broker itself is a standalone Web-Service that can serve both users and grid portals. The presented enhanced, adaptive scheduling solution with this Meta-Broker enables a higher level, interoperable brokering by utilizing existing resource brokers of different grid middleware. It gathers and utilizes meta-data about existing widely used brokers from various grid systems to establish an adaptive meta-brokering service. We have developed a new scheduling component for this Meta-Broker called Decision Maker that uses weighted functions with random generation to select a good performing broker to user jobs even under high uncertainty. We have evaluated the presented algorithms in a simulation environment based on GridSim with real workload samples. The presented evaluation results

affirm our expected utilization gains: the enhanced scheduling provided by the Decision Maker enables better adaptation and results in a more efficient job execution.

7 Bibliography

References

[1] Foster, I. and Kesselman, C. *Computational Grids, The Grid: Blueprint for a New Computing Infrastructure*, pp. 15–52, Morgan Kaufmann, 1998.

[2] Foster, I. and Kesselman, C. *The Globus project: A status report*, pp. 4–18, In Proc. of the Heterogeneous Computing Workshop, IEEE Computer Society Press, 1998.

[3] *EGEE middleware technical website*, http://egee-technical.web.cern.ch/egee-technical, September 2008.

[4] Erwin, D. W. and Snelling, D. F. *UNICORE: A Grid Computing Environment*, pp. 825–834, In Lecture Notes in Computer Science, volume 2150, Springer, 2001.

[5] *Job Submission Description Language (JSDL)*, http://www.ggf.org/documents/-GFD.56.pdf, September 2008.

[6] Kertész, A., Kacsuk, P., Rodero, I., Guim, F. and Corbalan, J. *Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management*, Technical report, TR-0116, Institute on Resource Management and Scheduling, CoreGRID – Network of Excellence, November 2007.

[7] Kertész, A. and Kacsuk, P. *Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service*, pp. 112–116, CoreGRID Workshop on Grid Middleware in conjunction with Euro-Par 2006, Dresden, Germany, LNCS, Vol. 4375, 2007.

[8] Kertész, A. and Kacsuk, P. *A Taxonomy of Grid Resource Brokers*, pp. 201–210, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2006), Springer US, 2007.

[9] *Open Grid Forum (OGF) website*, http://www.ogf.org, September 2008.

[10] *OGF Grid Scheduling Architecture Research Group*, https://forge.gridforum.org/sf/projects/gsa-rg, September 2008.

[11] Knuth, Donald E. *The Art of Computer Programming Volume 2.*, Section 3.2.1. Addison-Wesley Professional, 1997.

[12] Buyya, B. and Murshed, M. *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, pp. 1175–1220, Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, 2002.

[13] Howell, F. and McNab, R. *SimJava: A discrete event simulation library for Java*, In Proc. of the International Conference on Web-Based Modeling and Simulation, San Diego, USA, 1998.

[14] *Parallel Workloads Archive website*, http://www.cs.huji.ac.il/labs/parallel/workload, September 2008.

[15] Rodero, I., Guim, F. and Corbalan, J., Fong, L.L., Liu, Y.G. and Sadjadi, S.M. *Looking for an Evolution of Grid Scheduling: Meta-brokering*, Coregrid Workshop in Grid Middleware'07, Dresden, Germany, June 2007.

[16] Iosup, A., Epema, D.H.J., Tannenbaum, T., Farrellee, M. and Livny, M. *Inter-Operating Grids through Delegated MatchMaking*, In proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada, November 2007.

[17] Vazquez, T., Huedo, E., Montero, R. S. and Llorente, I. M. *Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures*, pp. 372–381, Euro-Par 2007, August 28, 2007.

[18] *The HPC-Europa Project website*, http://www.hpc-europa.org, September 2008.

Received September 10, 2008

C Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection

Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection

Anton Michlmayr, Florian Rosenberg, Philipp Leitner, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Wien, Austria
lastname@infosys.tuwien.ac.at

ABSTRACT

In service-oriented systems, Quality of Service represents an important issue which is often considered when selecting and composing services. For receiving up-to-date information, non-functional properties such as response time or availability can be continuously monitored using server- or client-side approaches. However, both approaches have strengths and weaknesses. In this paper, we present a framework that combines the advantages of client- and server-side QoS monitoring. It builds on event processing to inform interested subscribers of current QoS values and possible violations of Service Level Agreements. These events can trigger adaptive behavior such as hosting new service instances if the QoS is not as desired. We describe our QoS monitoring approach in detail, show how it was integrated into the VRESCo service runtime environment, and evaluate the accuracy of the presented monitoring techniques.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

QoS Measurement, Performance, Reliability

1. INTRODUCTION

In the past few years, Service-oriented Architecture (SOA) and Service-oriented Computing (SOC) [9] have emerged as a paradigm for addressing the complexities of distributed applications, and have finally gained acceptance from both industry and research. The overall idea is based on well-established standards for loose coupling and platform-independent interface descriptions. Web services represent the most common realization of SOA that build on the main standards SOAP and WSDL. Over time, several standards and specifications have been introduced for different issues in Web services research.

MW4SOC '09, November 30, 2009, Urbana Champaign, Illinois, USA

Quality of Service (QoS) plays a crucial role in service-oriented systems, for instance during service selection. When integrating external services into business processes, it is important to consider the quality guarantees of the service provider. Therefore, Service Level Agreements (SLAs) [4] are used to define the expected QoS between service consumer and service provider. In general, QoS attributes can be classified as deterministic or non-deterministic. The former indicates that the QoS attribute is known before a service is invoked (e.g., price, security, etc.), while the latter includes all attributes that are unknown at service invocation time (e.g., response time, availability, etc.). For non-deterministic QoS attributes, monitoring approaches can be used to continuously measure current QoS values.

Conceptually, there are two main approaches for QoS monitoring: Server-side monitoring is usually accurate but requires access to the actual service implementation which is not always possible. In contrast, client-side monitoring is independent of the service implementation but the measured values might not always be up-to-date since client-side monitoring is usually done by sending probe requests (i.e., test requests that are similar to real requests). In this paper, we aim at combining the advantages of both approaches which has been realized in the Vienna Runtime Environment for Service-oriented Computing (VRESCo) [8]. Therefore, we have linked an existing client-side QoS monitoring approach [14] together with server-side monitoring based on Windows Performance Counters [18]. Furthermore, event processing is used to integrate both approaches and provide means to monitor SLAs. If SLA obligations are violated, notifications are sent to interested subscribers using E-Mail or Web service notifications.

The contribution of this paper is threefold: Firstly, we present two approaches for monitoring QoS attributes of Web services. Secondly, we show how these approaches have been integrated into VRESCo, and how simple SLA obligations can be defined and monitored using event processing. Thirdly, we evaluate the accuracy of the two monitoring approaches and discuss how combining them leads to a comprehensive QoS monitoring framework.

The remainder of this paper is as follows. Section 2 shows our QoS model, and describes the client- and server-side monitoring approach. Section 3 introduces the VRESCo runtime, shows how QoS monitoring has been integrated and how SLA obligations are monitored. Section 4 presents the accuracy of the QoS monitoring approaches and briefly discusses the usefulness of combining them. Finally, Section 5 gives related work and Section 6 concludes the paper.

2. QOS MONITORING

In this section, we first briefly introduce the QoS model we have used in our work. Then we present two conceptually different monitoring approaches for Web services which we have integrated into VRESCo.

2.1 QoS Model

There are several definitions of QoS in literature [6,12,19]. Figure 1 depicts our QoS model which consists of the four categories *Performance*, *Dependability*, *Security/Trust* and *Cost/Payment*. In the remainder of this paper we focus on the first two categories since they can be measured automatically, with a special emphasis on those attributes that are currently measured by our approach.

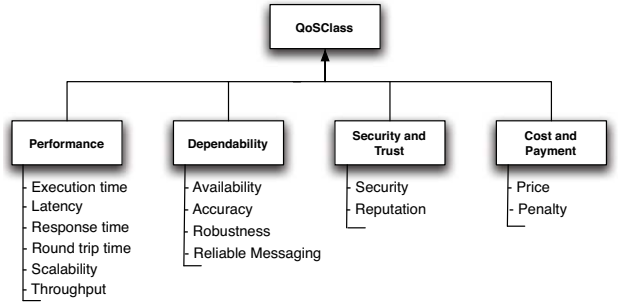


Figure 1: QoS Model

Performance-related attributes of services can be measured over several service invocations (see Figure 2). We consider the following attributes: *Execution time* q_{ex} represents the service invocation time at the provider. This consists of the actual *Processing time* q_{pt} and the *Wrapping time* q_w (i.e., time needed to wrap XML messages). *Latency* q_l defines the time needed for the client request to reach the service. *Response time* q_{rt} indicates the service invocation time at the service consumer (i.e., execution time plus latency), while *Round trip time* q_{rtt} measures the overall time needed for the request at the service consumer (i.e., response time plus wrapping time at the client). *Throughput* q_{tp} represents the number of service requests that can be processed within a given time period, while *Scalability* q_{sc} defines performance behavior of a service when the throughput increases. In general, these performance-related attributes are measured on the level of service operations.

Dependability-related attributes address the ability of services to avoid frequent and severe failures. In contrast to performance-related attributes, they are measured on the service-level. *Availability* q_{av} represents the probability that a service is up and running, while *Accuracy* q_{ac} defines the ratio of successful service executions in relation to the total number of requests. More details about our QoS model (including the formulas to calculate the attributes) can be found in [13].

2.2 Client-side Monitoring

The first approach to address QoS monitoring is done client-side using the QUATSCH tool [14]. The overall idea is to send probe requests to the services that should be monitored. The service invocation is thereby divided into the time periods shown in Figure 2 that correspond to the QoS attributes introduced above.

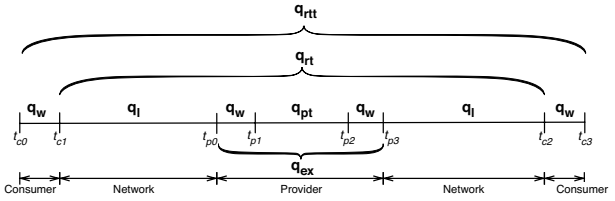


Figure 2: Service Invocation Times

The actual monitoring is done in three phases. In the preprocessing phase, the WSDL files of the services are parsed and stubs are generated. The performance measurement code is thereby weaved into the stubs using aspect-oriented programming (AOP). In the evaluation phase, the services are executed by probing arbitrary values as input parameters. If this is not successful, templates can be used to provide user-defined input. Finally, the result analysis phase stores the results of the evaluation phase in a database.

The interesting part of the client-side monitoring approach is that it is indeed able to accurately measure server-side attributes such as execution time. In QUATSCH, this is done using low-level TCP packet sniffing and analysis by leveraging the TCP handshake (i.e., SYN and ACK packets) to distinguish the different service invocation times [14].

2.3 Server-side Monitoring

The main drawback of the client-side approach is the fact that monitoring is done by regularly sending probe requests (e.g., every 5 minute). Single results should be handled with caution since they represent only snapshots (e.g., the service might be under heavy load when the probe request is sent). Decreasing the monitoring interval might mitigate this problem to some extent but this must be done carefully since short monitoring intervals (e.g., once every second) may finally affect the actual performance of the service.

Server-side monitoring addresses this problem by continuously measuring QoS attributes. Since no probe requests are needed anymore, the measured values represent "real" service invocations. However, as said above, this technique requires access to the actual Web service implementation which is not always possible in practice.

Windows Performance Counters (WPC), especially the counters regarding the Windows Communication Foundation (WCF) [10], are part of the .NET framework and provide such server-side QoS monitoring for Web services [18]. WPC supports a rich set of counters that can be measured at runtime. For our work, we focus on the following counters: *Call Duration* indicates the service invocation time which resembles *Execution Time* in the client-side approach. *Calls Per Second* defines how often a service has been invoked, while *Calls Failed Per Second* represents a similar counter for unsuccessful service invocations. Other performance counters (e.g., *Transactions Flowed Per Second*, *Security Validation and Authentication Failures*, etc.) could also be integrated seamlessly.

As before, monitoring is done in user-defined intervals. The different performance counter values are thereby aggregated and averaged within an interval, and finally re-set at the beginning of the next interval. For instance, if a service has been invoked 3 times, the average response time of these invocations is returned by the counter.

3. QOS/SLA INTEGRATION IN VRESCO

In this section, we show how the presented client- and server-side monitoring approaches have been integrated into VRESCo and how SLA monitoring can be achieved using the event processing engine.

3.1 VRESCo Overview

Before describing the QoS integration in detail, we briefly introduce the VRESCo service runtime environment [8]. The aim of this runtime is to address current SOC challenges and ease engineering of service-centric systems. More precisely, VRESCo addresses service metadata, QoS-aware service selection and service composition, dynamic binding and mediation of services, and complex event processing.

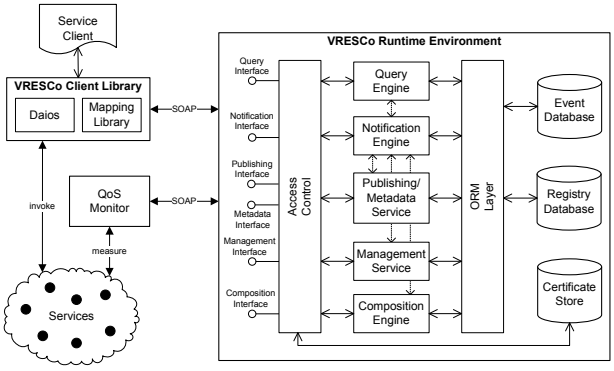


Figure 3: VRESCo Overview

The architecture is depicted in Figure 3. The runtime follows the "Software as a Service" concept and provides its core functionality as Web services that can be accessed using the client library or directly using SOAP. Services and associated metadata are published in the registry database using the publishing/metadata service, while the actual database access is done using an ORM layer. The VQL query engine provides type-safe querying of registry content, while the access control layer is responsible for allowing only authorized and authenticated users access to the VRESCo core services. The event notification engine [7] can be used to inform interested subscribers if events of interest occur (e.g., new service is published, etc.), while all events are additionally persisted in the event database. As discussed later, this event engine is leveraged for our QoS and SLA monitoring approach. Finally, the QoS monitor on the left-hand side represents the QUATSCH monitor [14] that follows the client-side monitoring approach. More information about the VRESCo core services and the service mediation approach are not within the scope of this paper and omitted for brevity. The interested reader is referred to [8].

3.2 QoS Integration

The overall architecture of our monitoring approach is shown in Figure 4. The client-side monitor QUATSCH was first integrated into VRESCo. Users can specify QoS monitoring schedules following the CRON time notation to define which service (or service operation) should be monitored in which time intervals. Since this is a client-side approach, the monitor runs on a dedicated QUATSCH host as shown in the middle of the figure. The actual monitoring is then done using AOP and TCP packet analysis as described in Section 2.2. Once the current QoS values have been measured, they are published into the VRESCo runtime. This is done via the QoS Manager that receives the values and, in turn, publishes them as corresponding QoS events into the event notification engine.

The monitoring is done regularly based on the user-defined monitoring schedules. The set of resulting QoS events represents the history of QoS information as collection of single QoS snapshots. To make these values easily accessible, they are aggregated by a QoS aggregation scheduler task on a regular basis, and finally attached to the corresponding service (or service operation).

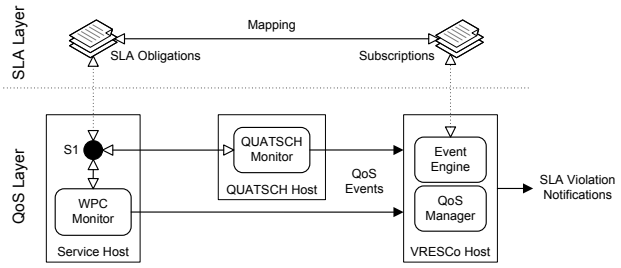


Figure 4: Monitoring Approach

As already discussed, the client-side approach has both strengths and weaknesses. Therefore, we decided to additionally integrate a server-side approach using WPC which is an integral part of the .NET framework. Consequently, the WPC-based approach is restricted to services implemented in .NET. The WPC monitor runs on the same host as the service (see Figure 4) and continuously monitors its QoS attributes. The measured values are published into the VRESCo runtime the same way as described before. For the WPC-based approach the monitoring schedules are defined in configuration files as shown in Listing 1. It defines which service/operation should be monitored, together with the monitoring and availability checking interval (in ms). The first describes how often the counters are retrieved while the latter is required since availability should be checked more often than other QoS attributes to get meaningful results.

Listing 1: WPC Monitoring Configuration

```
<vresco.qosmonitoring monitoringinterval="60000"
                      availabilitycheckinterval="5000">
 <webservices>
  <webservice wsdl="http://localhost:8013/s?wsdl">
   <operations>
    <add name="TestOperation"/>
   </operations>
  </webservice>
 </webservices>
</vresco.qosmonitoring>
```

In general, the two approaches are independent. However, some attributes can only be measured by one of the approaches (e.g., latency and response time have to be measured from the client-side). Table 1 shows the QoS attributes currently measured in VRESCo and depicts which approach has been taken for which attribute. *Throughput* and *Calls Per Second* seem to refer to the same QoS attribute. However, the first represents the maximum number of requests that can be processed, while the latter indicates the number of invocations that really occurred.

QoS Attributes	Monitored by
Execution Time	QUATSCH & WPC
Response Time	QUATSCH
Latency	QUATSCH
Availability	QUATSCH & WPC
Throughput	QUATSCH
Calls Per Second	WPC
Calls Failed Per Second	WPC

Table 1: QoS Attributes

It can be seen that two attributes are measured by both approaches. For both *Availability* and *Execution Time*, the WPC-based approach is usually more accurate since it does not need to send probe requests, but represents the values of real invocations. However, we decided to monitor using both approaches since the measured values might be different. In Section 4, we briefly discuss why this combination is useful and give some concrete examples.

3.3 SLA Monitoring

QoS monitoring approaches, as introduced in the last section, represent an essential foundation for SLAs, which define the expected QoS between service providers and consumers. In this section, we describe the SLA monitoring approach in VRESCo, and how clients can react to SLA violations. This approach is based on the VRESCo event engine and is depicted in the top part of Figure 4. To give a brief overview, simple SLA obligations can be attached to services. This is done using the publishing service that also allows to temporary start and stop SLA monitoring. These obligations are then transformed to subscriptions specified in the Esper Processing Language (EPL) [2] since the VRESCo event engine is based on the open source engine Esper. Those listeners can be directly attached to the engine, which does the actual matching between subscriptions and events. Finally, when such matches occur the subscribers are notified about the corresponding SLA violation.

Frameworks such as WSLA [4] have been proposed for defining complex SLAs, but they are rarely used in practice. Therefore, we decided to provide a mechanism for defining simple SLA obligations representing guarantees on the QoS attributes of services, which are shown in Table 2.

First of all, obligations can be either attached to service operations or revisions (in VRESCo, services can have multiple revisions). Every obligation is valid only within a given period of time after which it expires. The property name represents the QoS attribute to monitor (e.g., response time), while logical operator and property value are used to define threshold values (e.g., $<$ 500 ms). Aggregation functions (e.g., sum, max, avg, median, etc.) can further be defined on multiple QoS events. Obligations also define the notification mechanism and the address used for violation notifications (e.g., E-Mail or WS-Eventing notifications). Finally, sliding window operators can be used to define the time period to consider for the QoS events (e.g., the last 10 events or events within the last 5 minutes).

To give concrete examples, a simple SLA obligation could define that the availability of revision 23 should be greater than 0.99. This obligation is transformed to the following EPL expression (please note that logical operators must be inverted since subscriptions represent violation conditions):

Property	Description
Id	Identifier of the obligation
RevisionId	Identifier of the service revision
OperationId	Identifier of the service operation
Start Date	Start date of the obligation
End date	End date of the obligation
PropertyName	QoS attribute to monitor
Aggregation	Aggregation function on property
LogicalOperator	Logical operator used for comparison
PropertyValue	Threshold value used for comparison
ReactionType	Notification mechanism to use
ReactionAddress	Address of the subscriber
WindowType	Type of the sliding window operator
WindowValue	Value of the sliding window operator

Table 2: SLA Obligations

```
select * from QoSRevisionEvent where Revision.Id=23
and Property='Availability' and DoubleValue<=0,99
```

A more complex SLA obligation could define that operation 47 of service revision 11 should have an average response time of less than 500 ms within the last 24 hours. Besides the sliding window operator (`win:time`) this SLA obligation uses univariate statistics on event streams (`stat:uni` and `average`) which are provided by Esper:

```
select * from QoSOperationEvent(Revision.Id=11 and
Operation.Id=47 and Property='ResponseTime')\\
.win:time(24 hours).stat:uni('DoubleValue')
where average>=500
```

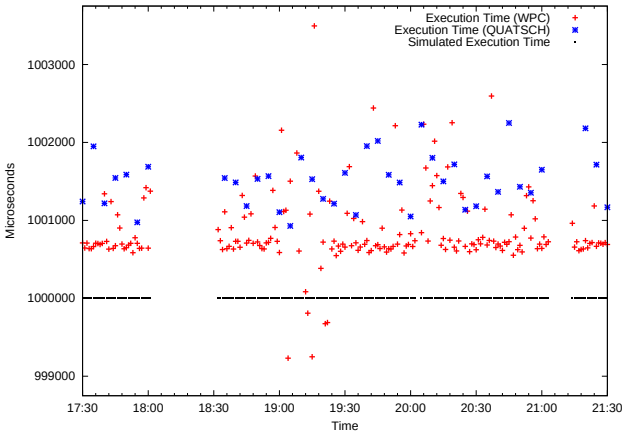
Once an SLA violation is detected, notifications are sent using E-Mail or Web service notifications to the specified address. The subscribers can then react accordingly, for instance by rebinding to functionally equal services [8]. In this regard, the SLA violation mechanism can also be used by service providers to monitor if services perform as intended. SLA violation notifications could then automatically trigger to start new instances of this service and publish them into VRESCo. Such scenarios and ways to define SLA penalty models are part of our future work.

4. EVALUATION

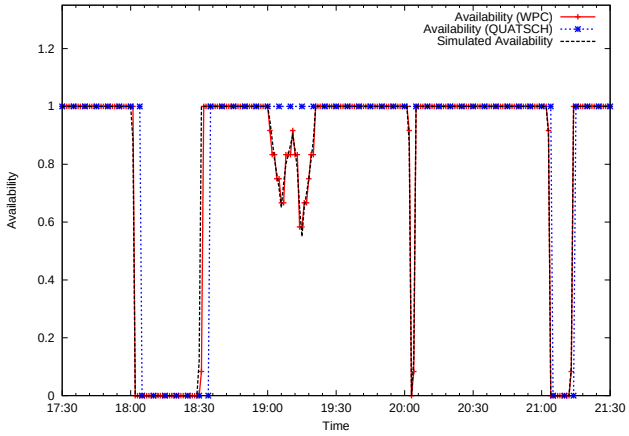
To evaluate our approach, we compare the accuracy of both monitoring approaches in terms of execution time and availability as two exemplary values that can be measured by both QUATSCH and WPC. Based on these findings, we discuss why a combination of both approaches is useful and highlight some of its advantages and disadvantages.

Our evaluation environment consists of a server hosting VRESCo and a set of C#/.NET dummy services that have a configurable execution time and a variable availability (3 downtimes of 30 min, 2 min and 10 min length, and simulated network problems with short interruptions between 19:00 and 19:20). Additionally, QUATSCH is hosted on a VMWare image running on a different server in the LAN.

Figure 5 depicts the results of our monitoring experiments where QUATSCH probes every 5 minutes whereas WPC measures every minute. We further use soapUI [16] to simulate clients. The different measurement intervals are based on the fact that QUATSCH sends real invocations to probe a service, while WPC has lower overhead because it queries performance counters provided by the operating system.



(a) Execution Time



(b) Availability

Figure 5: Accuracy of WPC and QUATSCH

Figure 5(a) depicts the measured execution time over 4 hours. The results show that both approaches are pretty accurate. The deviation from the simulated execution time (1 sec) is less than 2 ms for most measurements. The values for QUATSCH indicate that execution time can be indeed measured from the client-side. Additionally, it can be seen that WPC is more accurate because it represents the average execution time of real invocations. The gap between simulated execution time and average value measured by WPC is 0.88 ms which is partly caused by internal processing of the test services (e.g., threading, console output, etc.).

Figure 5(b) shows the simulated and measured availability of the test services. It can be observed that WPC detects downtimes faster than QUATSCH, which is due to the shorter monitoring interval. WPC further divides this interval into availability checking intervals (5 sec). Therefore, the availability of one monitoring interval can also be measured (e.g., at 18:31 and 20:02). In contrast to that, QUATSCH cannot do this fine-grained distinction (i.e., availability is either 0 or 1). As a result, QUATSCH does not recognize the short downtime at 20:02. The same is true for the timespan between 19:00 and 19:20 where WPC is quite accurate whereas QUATSCH does not detect this at all. It should be noted that the same behavior can be observed when the execution time in Figure 5(a) is varying.

Nonetheless, combining both approaches is still useful. Firstly, some QoS attributes can only be measured from the client-side (e.g., latency). Secondly, it is possible to distinguish between client- and server-side view on some QoS attributes (e.g., availability). For instance, if there is no network connection on the QUATSCH monitoring host, client-side availability decreases even if the service is running. However, this can be verified by the WPC approach. Thirdly, bogus server-side measurements can be detected by the client-side approach, by comparing measured QoS values over a longer period of time. Fourthly, another dimension to client-side monitoring could be added by integrating actually perceived QoS values on the client-side (in addition to the measured values of the QUATSCH probe requests). However, the combination of both approaches also has some drawbacks. For instance, clients must agree to install monitoring software which may not always be the case.

Finally, we have shown that the accuracy of the monitoring approaches makes them suitable for SLA monitoring as introduced in Section 3.3. As shown in our previous work [7], the throughput of the VRESCo event engine is high enough for the expected number of services (e.g., 2000 services with the same monitoring intervals as above). Furthermore, since SLA monitoring is based on events, it is easily possible to subscribe to SLA violations and react adaptively if needed.

5. RELATED WORK

Several different QoS models have been proposed in literature (e.g., [6, 12, 19]). However, most approaches do not discuss how QoS can be monitored. An overview of QoS monitoring approaches for Web services is presented by Thio et al. [17]. The authors discuss various techniques such as low-level sniffing or proxy-based solutions. The prototype system presented in their paper adopts an approach where the SOAP engine library is instrumented with logging statements to emit the necessary information for QoS measurement. A major drawback of this approach is the dependency on the modified SOAP library and the resulting maintenance and distribution of the modified library.

QoS monitoring has been an active research area for years which is not only focused on Web service technology. For instance, Garg et al. [3] present the WebMon system that aims at monitoring the performance of web transactions using a sensor-collector architecture. Similar to our work, their approach correlates client- and server-side response times which are measured by different components. In their work, the question is whether to instrument the web server or the web browser for doing the performance measurements.

There are many existing approaches for SLA monitoring and violation detection (e.g., [1,5,11,15] just to name a few). Skene et al. [15] introduce SLAng which is a general SLA language not only focused on Web services, but targeted to distributed systems and applications with reliable QoS characteristics. The language is modeled in UML while the syntax is defined using XML schema. The authors further define a model for all parties and services involved in such agreement. The actual constraints in the SLAs are then defined using the Object Constraint Language (OCL).

Raimondi et al. [11] describe an SLA monitoring system that translates timeliness constraints such as latency or availability of SLAs into timed automata, which are then used to verify execution traces of services. Their approach uses SLAng for defining SLAs and is realized as Axis handler.

Lodi et al. [5] describe a middleware that enables SLA-driven clustering of QoS-aware application servers. Instead of existing standards, they use XML for defining SLAs which was inspired by SLAng. The architecture consists of three components: The Configuration Service is responsible for managing the QoS-aware cluster, the Monitoring Service observes the application at runtime to detect violations of SLAs, and the Load Balancing Service intercepts client requests to balance them among different cluster nodes. If the cluster is mainly idle or close to breach the SLA (e.g., the response time converges to the upper bound), it is reconfigured (i.e., add/release nodes).

Chau et al. [1] present a similar approach for modeling and event-based monitoring of SLAs which is part of the eQoSystem project. The SLA model refines the WSLA specification: SLAs consist of multiple SLOs and use various metrics that indicate different measurement aspects of a process. Furthermore, action handlers can be defined to react when SLOs are violated. Similar to our work, the SLA monitoring approach is based on events. These events are assumed to be emitted by the business process and contain a snapshot of the current process state. In contrast to that, our QoS events focus on the service- and operation-level. Furthermore, we additionally address how QoS attributes of Web services can be monitored from client- and server-side.

6. CONCLUSION

Monitoring QoS attributes of Web services is an essential aspect to enforce SLAs established among business partners. In this paper, we have shown that a combination of client- and server-side QoS monitoring can be beneficial regarding the overall monitoring capabilities since both approaches have strengths and weaknesses. These monitoring capabilities combined with a powerful Web service runtime enable an event-based detection of SLA violations for Web services, while subscribers can react appropriately to such violations. For future work, we plan to automatically react to SLA violations, such as deploying new service instances on-the-fly or dynamically increase certain virtual machine capabilities (that are often used to host Web services). Furthermore, we envision to measure and publish the actual response times at the client-side, in addition to the QUATSCH measurements that rely on probe requests.

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 Alexander Schindler for realizing the WPC-based monitoring approach.

7. REFERENCES

[1] T. Chau, V. Muthusamy, H.-A. Jacobsen, E. Litani, A. Chan, and P. Coulthard. Automating SLA Modeling. In *Proc. of the 2008 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'08)*, 2008.

[2] Esper, 2009. `http://esper.codehaus.org/`.

[3] P. K. Garg, K. Eshghi, T. Gschwind, B. R. Haverkort, and K. Wolter. Enabling Network Caching of Dynamic Web Objects. In *Proc. of the 12th Int. Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, 2002.

[4] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[5] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Transactions on Software Engineering*, 33(3):186–197, 2007.

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

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

[8] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. Technical report, Vienna University of Technology, 2009. `http://www.infosys.tuwien.ac.at/Staff/michlmayr/papers/TUV-1841-2009-03.pdf`.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.

[10] C. Peiris, D. Mulder, A. Bahree, A. Chopra, S. Cicoria, and N. Pathak. *Pro WCF: Practical Microsoft SOA Implementation*. Apress, 2007.

[11] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service SLAs. In *Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering (SIGSOFT'08/FSE-16)*, 2008.

[12] S. Ran. A Model for Web Services Discovery with QoS. *SIGecom Exchanges*, 4(1):1–10, 2003.

[13] F. Rosenberg. *QoS-Aware Composition of Adaptive Service-Oriented Systems*. PhD thesis, Vienna University of Technology, June 2009.

[14] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proc. of the IEEE Int. Conference on Web Services (ICWS'06)*, Sept. 2006.

[15] J. Skene, D. D. Lamanna, and W. Emmerich. Precise Service Level Agreements. In *Proc. of the 26th Int. Conference on Software Engineering (ICSE'04)*, 2004.

[16] soapUI, 2009. `http://www.soapui.org/`.

[17] N. Thio and S. Karunasekera. Automatic measurement of a QoS metric for Web service recommendation. In *Proc. of the Australian Software Engineering Conference (ASWEC'05)*, 2005.

[18] WCF Performance Counters, 2009. `http://msdn.microsoft.com/en-us/library/ms735098.aspx`.

[19] 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*, 30(5):311–327, May 2004.

D Selecting Web Services Based on Past User Experiences

Selecting Web Services Based on Past User Experiences

Philipp Leitner, Anton Michlmayr, Florian Rosenberg, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract

Since the Internet of Services (IoS) is becoming reality, there is an inherent need for novel service selection mechanisms, which work in spite of large numbers of alternative services and take the user-centric nature of services in the IoS into account. One way to do this is to incorporate feedback from previous service users. However, practical issues such as trust aspects, interaction contexts or synonymous feedbacks have to be taken into account. In this paper we discuss a service selection mechanism which makes use of structured and unstructured feedback to capture the Quality of Experience that services have provided in the past. We have implemented our approach within the SOA runtime VRESCo*, where we use freeform tags for unstructured and numerical ratings for structured user feedback. We discuss the general process of feedback-based service selection, and explain how the problems described above can be tackled. We conclude the paper with an illustrative case study and discussion of the presented ideas.*

1. Introduction

The term "Internet of Services" [1] (or IoS for short) describes the unification of Web 2.0, Service-Oriented Architectures [2] (SOA), Ubiquitous and Pervasive Computing, Semantic Web Services and Mobile Computing that we are currently experiencing. Soriano et al. define the IoS as "the notion of a global, user-centric SOA enabling a real Web of Services made up of a global mesh of interoperable user-centric services" [3]. In such a global network of human-provided and human-consumed services there is an evident need for mechanisms which allow (human) users to select services from a vast number of alternatives. Besides actual functionality, the main factor influencing the service selection process is the Quality of Service (QoS) of the alternatives. However, in addition to the software parts, services in the IoS often also contain a physical part, which may be as or even more important to the user than the technical service itself. For example, considering a hotel booking service (which we will use as a running example in the remainder of this paper), we can safely assume that for the majority of users the technical booking service is of much less relevance for the overall satisfaction with the business transaction than the quality of the hotel itself. Consequently, typical fine-grained QoS metrics such as service response time are not enough to capture the full user experience in the IoS. Instead, the user needs quality metrics which describe the quality of the business transactions in an end-to-end fashion. We refer to this less formalized end-to-end quality metric as Quality of Experience (QoE [4]).

The problem of QoE is that it is inherently hard to quantify and measure automatically because of the physical part mentioned above. However, the Web 2.0 trend of folksonomies [5] provides a reasonable way to capture QoE by incorporating human factors, and evaluate the quality of services based on feedback from earlier transactions, e.g., using numerical ratings. Additionally, folksonomies often make use of the concept of emergent semantics [6] (i.e., determining the semantics of resources through large amounts of independent, publicly available metadata, such as tags) to functionally describe resources.

The main contribution of this paper is the combination of these two concepts into a service selection framework for the IoS, which captures human-perceived QoE using both freeform tags (unstructured feedback similar to Web 2.0 tags) and numerical ratings (structured feedback, i.e., ratings) to narrow down and rank available service alternatives. Unlike current systems, unstructured feedback is used to describe the satisfaction of users with a service instead of its semantics. We explain how this leads to practical challenges such as tag merging, preventing

spamming of service providers or including the context of invocations, and show how these issues can be tackled. Furthermore, we explain a system implementation based on the VRESCo [7] SOA runtime which realizes these ideas.

The remainder of this paper is structured as follows. Section 2 gives an overview over the most important related work. In Section 3 we introduce our approach to feedback-based service selection. In Section 4 we explain how the concepts presented in this paper are integrated into the VRESCo SOA runtime. Section 5 describes the implementation of our prototype tool, which is discussed in Section 6. Finally, some concluding remarks and an outlook on our future work are provided in Section 7.

2. Related Work

Web service discovery based on QoS and service semantics has been a hot topic in research for some time [8], [9]. Generally, these approaches have a similar scope to our work in that they aim at allowing users to find not just any service to fulfill a task, but the best one. However, as argued in Section 1, we strongly feel that QoS alone is too narrow to fully represent the experience of service users in the IoS. The idea that technical QoS parameters such as availability and response time are too limited to capture the experience provided to end users has originally been introduced by van Moorsel in [4]. This work also concluded that "the relationship between system quality, user experience (...) is hard to make concrete", however, no ideas are presented on how QoE can be estimated in advance, which is essentially the scope of this paper. Much research has already been conducted in the area of collaborative tagging systems, e.g., in [10]. These works do not apply the concepts of tagging to the area of services engineering, but focus on general Web 2.0 systems such as Del.icio.us[1]. However, these earlier works provided many helpful insights which helped shape the contributions in this paper. For instance, [10] was first to identify that tagging is often used not only to classify information, but also to rate it, a notion that we also use throughout our work.

Combining Web 2.0 and SOA has been explored in [11]. This work introduces the concepts of service communities, and discusses a prototype middleware to support Web 2.0 aspects such as tagging. Unfortunately, little technical detail about this prototype is openly available. Generally, our work bears some similarities to the idea of discovering service semantics through tagging [12]. A conceptually similar approach is implemented in SOALive [13]. SOALive is a system for describing and discovering situational enterprise services based tag clouds, which is similar to the work we present. The main difference of our work and these approaches is that we use tagging for an entirely different purpose – in these works tagging is used to describe what a service does, and not how well it performs. In our work, the "what" aspect is handled using the VRESCo metadata model, while feedback is used purely to evaluate "how well" a service performs.

The idea of using past transactions for suggestion is the foundation of recommender systems [14]. In a recommender system people provide recommendations as input, which the system aggregates and transforms to support the selection of others. At the core, our system is a recommender system for IoS services, therefore, our work faces similar problems. One of these problems is "free-riding" users, i.e., users which consume feedback from others while not contributing themselves. Another one is the "vote early and often" phenomenon [14], i.e., service owners spamming the system by providing massive amount of negative feedback for competitors and/or positive feedback for their services. In our approach, we use the concepts of trust to deal with the latter issue. The former problem is not addressed directly, however, we argue that free-riding becomes more and more irrelevant with increasing scale, as demonstrated daily by many Web 2.0 platforms which work perfectly on the notion of voluntary contributions (e.g., Wikipedia[2]). An early recommender system for Web service selection has been presented in [15]. However, this system is only based on numerical ratings and seems therefore limited. A more advanced Web service recommender system has been presented in [16]. The IC-Service system presented there actively monitors Web service usage (using a remote client which has to be incorporated into client applications) and suggests Web services based on how other applications use services in similar contexts. The main difference between this work and ours is that they focus purely on implicit feedback (i.e., information that their client application collects), while we use a more explicit approach where users actively contribute feedback through tags and ratings. A system based on explicit feedback is discussed in [17], where explicit structured feedback from clients is used to evaluate if service providers violate their Service Level Agreements (SLAs), and an incentive system is presented which motivates clients to report truthfully on their perceived service quality.

In the database community the Skyline operator [18] is a well-known concept to find the most interesting alternatives in a large set of data points. This is related to our work, however, the Skyline approach relies on a well-structured set of dimensions to rank data (i.e., in order to use the Skyline approach all services need to be

1. http://delicious.com/

2. http://www.wikipedia.org/

ranked according to similar features). This is in line with traditional QoS models, however, we argue that the IoS is inherently less structured. In such a world the Skyline approach is, unlike our work, not applicable.

3. Solution Outline

Feedback-based service selection considers the selection of services from a large number of alternatives based on existing feedback from past user transactions. In our approach we use two different types of feedback. Firstly, we make use of unstructured feedback given as any number of freeform strings (tags). This type of feedback is used to characterize the QoE provided by a service in a very open form. Secondly, we use numerical ratings between 1 and 5 as structured feedback to numerically rank services for selection.

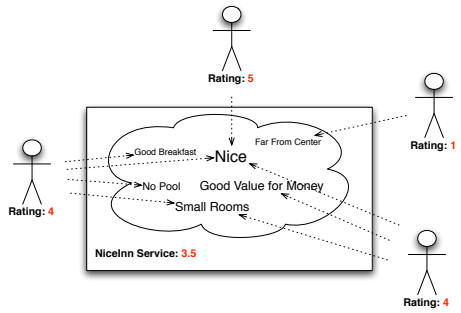


Figure 1. Illustrative Example

Figure 1 depicts the type of feedback used in this paper for an imaginary hotel booking service (NiceInn). There is feedback from four different customers for the service. The average numerical rating is *3.5* (on a scale from 1 to 5, with 5 being the best). However, much more interesting is the unstructured feedback associated with the service (visualized as tag cloud). From these tags more detailed information about the service can be learned, such as that three of four customers seemed rather satisfied, but the rooms were considered too small, and there is no pool. Clearly, these tags transport more fine-grained quality information. Some of this additional information may be unimportant for some future clients, but essential for others (e.g., if a client searches for a hotel for a business trip she may not care about there being a pool in the hotel, however, for a client who wants to go on summer vacation the pool might be a must-have). Note that in our model these tags are used to capture quality rather than semantics or functionality, even though some tags (like `No Pool` in the example) could also be considered functional descriptions.

The high-level process of feedback-based service selection is depicted in Figure 2. Generally, the process is a positive feedback loop, i.e., successful executions of the service selection process feed back into the system, which in turn (in tendency) improve the overall performance of the system. The selection process has five stages, which are executed sequentially.

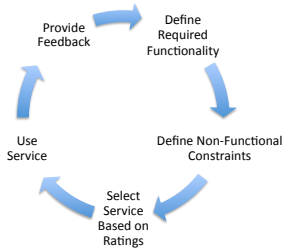


Figure 2. Service Selection Cycle

Firstly, the needed functionality is defined (e.g., being able to book a hotel). This is outside the scope of this paper. In our solution we mainly use the means provided by the VRESCo system, however, other approaches such as Semantic Web Services would be suitable as well. Secondly, in order to reduce the number of alternatives presented, non-functional constraints are defined. This is done by defining which tags should and should not be associated with the service to find (e.g., services should have the tags `Nice` and `Good Breakfast` associated, but not `No Pool`). Logically, this step is used to pre-filter the number of alternative services using hard constraints on the QoE of the service. Thirdly, a concrete service is picked from the remaining alternatives. This step is supported using both the structured and unstructured feedback: services are ordered based on average ratings given by users, and tag clouds can be used to quickly get an impression of what the general opinion of past users about the service was, besides the hard constraints formulated in the second step. When a concrete service is selected, it can finally be consumed. Lastly, the user should herself contribute back to the system and provide both tags and numerical rating to characterize the perceived quality of the service. In the following section we will explain how this cycle is implemented and supported in the VRESCo system, and how the challenges evident in such systems can be tackled.

4. Feedback Metadata and Extensions

We have integrated our feedback-based service selection model into the VRESCo (Vienna Runtime Environment for Service-Oriented Computing) SOA runtime environment. Details about VRESCo can be found elsewhere [19], and will be omitted here due to space restrictions. In the following we will present how the metadata necessary for feedback-based service discovery have been

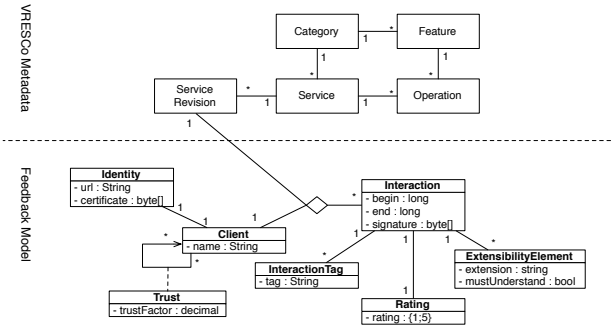


Figure 3. Service Feedback Metadata Model

implemented (grounded mostly on work presented in [20]), and which extensions have been provided in order to deal with real-world issues.

4.1. Basic Metadata Model

In [20], we have presented the general metadata model for the VRESCo infrastructure. This paper introduced the notion of *Feature*s and *Categories*, which are used to represent and group activities in a domain, e.g., booking a hotel. In VRESCo, features are semantically well-defined activities in the domain, with clearly defined inputs and outputs. In this paper we use *Feature*s to capture the functional aspects of service discovery, i.e., the first step in the cycle in Figure 2 is implemented by selecting a concrete *Feature* from the VRESCo metadata model.

Figure 3 shows an outtake of the original VRESCo metadata model ("VRESCo Metadata"), and the new data items that we have added in order to capture structured and unstructured user feedback ("Feedback Model"). Note that we have simplified the model for clarity in this figure. At the core, the idea is to capture *Interaction*s of *Service Client*s with concrete *Service Revision*s. For every *Interaction* we store exactly one *Rating* and any number of *Interaction Tags*. In our model we use the *Rating* as structured feedback with defined semantics (i.e., the rating is always a value between 1 and 5, 5 again being the best), while the *Interaction Tags* serve as means for the client to express additional feedback on the *Interaction* in a less structured way. Any additional information can be expressed using *Extensibility Elements*, which may contain arbitrary information, usually encoded in XML. Extensibility elements are (unlike rating and interaction tags) not standardized, i.e., usually not every extensibility element is interpretable for every client. The boolean "mustUnderstand" flag is used to indicate how a client should proceed when she receives an interaction with an extensibility element that she cannot interpret. If the flag is set to "true" the client ignores the interaction as a whole, if it is set to "false" the client ignores the extensibility element. Finally, begin and end date of the interaction are stored (as UNIX timestamp) to allow clients to e.g., only consider the more recent feedback.

4.2. Extensions

Using the model described above we are able to represent the basic structured and unstructured feedback necessary to implement a service selection process such as the one described in Section 3. However, there are a number of challenges associated with this basic model:

- Firstly, tagging systems such as the one described above face the problem of redundancy in tags, i.e., many different tags with very similar or identical semantics (e.g., `Nice`, `Good`, `Very Well`), which is further amplified by varying spellings and typing errors. In collaborative tagging systems the idea of tag merging based on tag similarity has been proposed to overcome this issue [21]. We provide two different merging strategies (based on co-occurence and cosine similarity) to define similarity of tags.
- Secondly, tagging systems are known to be susceptible to spamming of service providers [6]. We use a trust model to represent the reliability of a feedback source, in order to mitigate the influence of spam. On this account it is also necessary to provide means to verify the integrity of feedback (i.e., to verify that the feedback has not been forged).
- Finally, the above model ignores the context [22] in which a service has been used when it has been rated. This may be problematic for services which are often used in varying contexts, and exhibit a different QoE depending on it (e.g., the QoE provided by a hotel service may be totally different depending on whether the service is used in the context of a business or leisure trip). Therefore, we allow to annotate feedback with context information, so that future users may choose to incorporate only feedback which has been collected in equivalent circumstances of usage.

In the following we will discuss our solutions to these problems in more detail.

4.2.1. Tag Merging. To define similarity of freeform tags a number of possibilities exist [23]. Most of these similarity measures focus on how often tags have been used to describe the same service (co-occurence) or on the similarity of their usage patterns (cosine similarity). In VRESCo, we allow system administrators to select from three different tag merging strategies. The most basic one is the $none$ strategy (no merging at all). *Co-occurence merging* considers all tags t_1, t_2 to be similar which have a

normalized co-occurce ($nfreq(t_1, t_2)$) greater than a user-defined threshold. The co-occurence of two tags t_1 and t_2 is defined as the number of times t_1 and t_2 have been associated with the same service. We use the normalized co-occurence (co-occurence divided by the total number of times t_1 and t_2 have been used), because it is in $[0; 1]$, with $nfreq(t_1, t_2) = 0$ meaning that the tags are never used together, and $nfreq(t_1, t_2) = 1$ meaning that the tags are only used in conjunction. Finally, *cosine similarity merging* considers tags to be similar if they have a cosine similarity $cossim(t_1, t_2)$ greater than a given threshold. We use the definition of cosine similarity presented in [21]:

$$cossim(t_1, t_2) := \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|} \quad (1)$$

v_i are vectors, with $v_{t,t'} := freq(t, t')$ (i.e., every vector v_i consists of the non-normalized distances to all tags). $v_{t,t}$ (the distance of a tag to itself) is defined as 0. $\|v_1\|$ denotes the magnitude of v_1.

The actual process of tag merging is implemented using the following algorithm: for each tag we check if we should merge it with any other tag in the database (i.e., if the similarity of those tags is higher than a given threshold), and if both tags have been used more than once (we explicitly exclude tags which have been used only once, since we have too little information to merge these tags). If this is the case we update all occurrences of the tag which has been used less frequently with the more popular one. After merging a tag, we recalculate its similarity to every other tag. Consider for example the case of co-occurence merging with a threshold of 0.2. In this case, the two tags `Good` and `Very Nice` are merged if and only if they are used together (i.e., describing the same service) at least 20% of the time.

4.2.2. Trust Relationships. Another problem is spamming of service providers. One approach to handle this problem is to consider only feedback from trusted other users. However, this would arguably hamper the usefulness of the system, since too little feedback would be available to most services. One compromise to prevent spamming on the one hand and still incorporate a reasonable amount of feedback is to propagate trust relationships in a "friend of a friend" type of way.

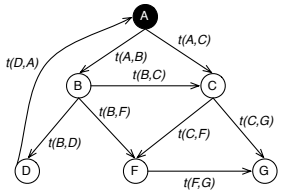


Figure 4. Example Trust Graph

Trust relationships between users in VRESCo are weighted using a "trust factor", a decimal value in $[0; 1]$, where 0 represents "no trust" and 1 "full trust". Inherently, the trust relationships between *Service Client*s form a weighted, directed graph, which we refer to as trust graph. Figure 4 sketches a small example trust graph, in which trust is propagated using a simple algorithm: the trust between two clients is the weighted length of the unweighted shortest path (i.e., the path with the smallest number of edges, ignoring edge weights) between these two clients in the trust graph; if two or more equally long shortest paths exist the one yielding a lower trust is used. Client *A* trusts *B* with a trust factor of $t(A, B)$, *C* with a trust factor of $t(A, C)$ and *F* with either $t(A, B) \cdot t(B, F)$ or $t(A, C) \cdot t(C, F)$, depending on which is less. Note that clients can explicitly assign a trust factor of 0 to essentially block a user. Clients choose a threshold of trust factor up to which they consider a client still trustworthy. Greater trust thresholds prevent spamming with higher certainty, however, this also leads to fewer interactions being used for selection, decreasing the value of the feedback-based service selection in general.

Trust relationships as described above inherently rely on the system being able to guarantee that feedback has not been forged (i.e., that a given feedback is indeed from the user that it claims to be from, and that the feedback has not been changed). We use digital signatures and OpenID[3] to let users explicitly verify the feedback that they use for their selection.

4.2.3. Context Information. The feedback metadata model as depicted in Figure 3 allows for *Extensibility Element*s, which incorporate more information about *Interaction*s than the basic model of tags and rating. One concrete usage of *Extensibility Element*s in our system is to capture the context of *Interaction*s. In our current prototype we re-use the context model presented in [24] for the VieCAR framework. In this model *Activities* can be modeled, which represent the context in which action has been carried out, e.g., booking a leisure trip. This context information may be used to filter out *Interaction*s which are less relevant, e.g., if a client is searching for a hotel service for a leisure trip interactions which have been conducted in a leisure context are more likely relevant to her than interactions in a business context.

For our prototype we have predefined a number of different contexts for the travel case study as described in Section 3. Currently, users can only select a context from these provider-defined activities. Clearly, these predefined contexts are scenario-dependent. As part of our future work we therefore plan to improve on this facet by allowing

3. http://openid.net/

users to define their own contexts and how they relate to existing ones, e.g., by refining existing contexts.

5. Prototype

We have implemented an end-to-end prototype system as an ASP.NET application on top of the VRESCo SOA runtime environment. The system is implemented in the C# programming language, using .NET 3.5 and the Windows Communication Foundation[4] (WCF) framework.

Our application supports the service selection cycle as described in Figure 2, i.e., the usage of the tool follows the five central steps "Defined Functionality", "Define Constraints", "Select Service", "Use Service" and "Provide Feedback". Required functionality can be selected using the VRESCo construct of *Feature*s. Non-functional constraints are defined by selecting tags describing qualities that the service should or should not exhibit, while the actual selection of a service is based on the rating given by previous users. Finally, the system allows to invoke a service and provide feedback. Note that obviously the system supports only the invocation of the "technical" part of a service, e.g., in the hotel scenario our application supports the invocation of the hotel booking service, but the feedback provided by the user should be based not only on her experience with the booking service, but with the hotel stay in general. Therefore, the last step "Provide Feedback" may be executed significantly after the actual technical invocation of the Web service.

Generally, the system makes heavy use of tag clouds. Tag clouds are used to visualize unstructured user feedback and to support `Feature` selection. For the actual invocation of selected Web services the prototype incorporates the Dynvoker [25] framework, which is able to invoke both SOAP-based and RESTful Web services. Dynvoker allows to generate easy-to-use graphical user interfaces from WSDL or WADL service descriptions. In Figure 5 we have depicted the interfaces to select non-functional properties using tag clouds (Figure 5(a)) and to select services using structured feedback (Figure 5(b)). Users can select tags by dragging-and-dropping tags from the tag cloud onto the "+" and "-" icons. Furthermore, our prototype allows to manage user trust relationships (as described in Section 4), by defining "buddies" and blocking users. Tag merging following one of the three tag merging strategies described in Section 4 has been implemented as a server-side background process. This is because tag merging is computationally expensive, and cannot be done dynamically at request time. Therefore, the tag merging strategy used cannot be selected by the user of the system, but needs to be defined globally by a system administrator.

4. http://msdn.microsoft.com/en-us/netframework/aa663324.aspx

6. Case Study

To illustrate the usefulness of our approach we will now exemplify the usage of our system based on an illustrative case study. For this, we will use feedback on hotels gathered from TripAdvisor[5]. We collected feedback on 40 popular hotels located in Vienna, which includes data about roughly 1750 interactions.

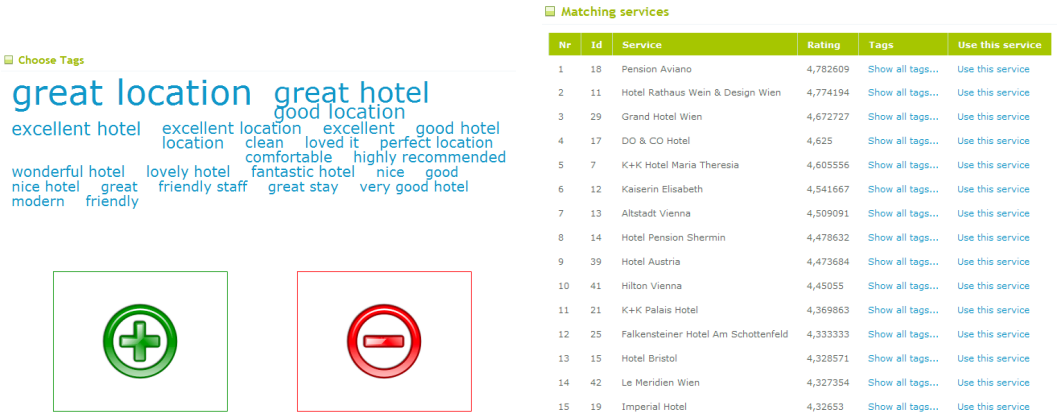
Figure 5(b) shows the top 15 hotels ranked according to their average numerical rating. This is akin to the traditional Web 2.0 way of listing services based on reviews. However, we can see that the difference of the average rating among the top 10 is only marginal. From this numerical ranking alone it is hard to judge which hotel is actually the best to choose. We can now use the unstructured feedback provided by previous users to get a clearer picture. Figure 5(a) shows the tag cloud visualization of the available feedback to the hotels. We have now a more detailed control over which properties we would like the service to exhibit. For instance, we are able to define that a good location is important for us, and that we want the hotel staff to be very friendly. These properties are specified by selecting the tags `good location` and `friendly staff` as required. The list of matching hotels is now significantly shortened to 5 (arguably more relevant) services. Additionally, among these 5 results, the difference in rating is more significant, allowing us to make a more informed decision.

Merging Strategy	Threshold	# Distinct Tags
None	–	1666
Co-Occurence	0.45	1590
Co-Occurence	0.20	1479
Co-Occurence	0.15	1469
Cosine Similarity	0.003	1591
Cosine Similarity	0.001	1503
Cosine Similarity	0.0005	1483

Table 1. Distinct Tags Per Merging Strategy

However, as we can see in Figure 5(a), there are currently many tags with very similar semantics (e.g., `great location`, `excellent location` and `good location`), which are blurring the picture. This can be resolved to some degree by enabling tag merging. In Table 1 we have summarized how various tag merging configurations influence the total number of tags contained in the system. Note that the thresholds used for cosine similarity are much lower than for co-occurence merging. This is due to the fact that cosine similarity values are by definition much lower than the co-occurence values (the arithmetic mean of all co-occurence similarities in our data set is 0.0532, and only 0.0011 for cosine similarity). From Table 1 we can see that cosine similarity merging

5. http://www.tripadvisor.com/



(a) Non-Functional Property Selection Using Tag Clouds

(b) Services Ranked by Structured Feedback

Figure 5. Prototype Screenshots

is in tendency more "conservative", in the sense that tags are merged less frequently. In our case study using co-occurence merging with a threshold of 0.25 produces the most useful results, leading to a rather clear tag cloud (see Figure 6). For reasons of brevity we do not discuss the consequences of including context information and trust in this illustrative example.

Figure 6. Tag Cloud After Merging

To draw some conclusions from this illustrative example, we can state that the general ideas of this paper seem valid when applied to real-world data. We argue that our approach significantly improves selection quality as compared to e.g., UDDI specifically in scenarios where the quality of services is not easily described using typical QoS metrics such as response time alone. However, two limitations should not be left undiscussed. Firstly, it is obvious that our way of filtering for non-functional properties is heuristic: just because no user ever used the tag `clean` or a synonym to describe a hotel it does not necessarily mean that it is dirty. However, this problem decreases with an increasing number of interactions stored, which means that our approach inherently relies on a certain scale of the system (regarding number of users and service alternatives). Secondly, when examining our example data set it became obvious that most user feedbacks are rather positive. In our sample, only about $\frac{1}{10}$ of the tags can be considered negative. After tag merging, some negative tags are contained, but they are still relatively scarce as compared to the positive ones.

7. Conclusion

The IoS is a global service ecosystem, with the two most prominent features being the large scale and the strong involvement of humans, also as service clients. Therefore, novel service discovery methods are necessary which take these characteristics into account. In this paper we have presented one approach which satisfies these requirements. We have explained the notion of QoE, and how QoE can be modeled using feedback from past user transactions. In VRESCo we have integrated a feedback-based service selection mechanism, which uses simple string tags to capture unstructured user feedback, and numerical ratings for structured feedback. Additionally, we use a trust model to prevent service providers from spamming the system, and digital signatures to verify feedback integrity. The context of interactions of users and services is incorporated using an existing context model. Finally, we allow to merge tags with similar semantics. We detailed the implementation of an end-to-end prototype system, and explained some aspects of the system based on an illustrative example.

Our current implementation can be seen as a first step towards a full selection infrastructure. To that end future work has to be carried out in some directions. Firstly, we plan to extend our model to incorporate implicit user feedback besides the explicit feedback discussed here. Such implicit feedback includes monitoring of user behavior and preferences. Secondly, more thought needs to be put into the way context information is used. Currently, contexts can either be "equivalent" or "not equivalent", ignoring the possibility of partially matching contexts. Finally, even though the first experimental results as discussed in Section 6 are promising we still need to conduct some real-life experimentation to further validate the ideas presented.

This is especially relevant to further "fine-tune" the various parameters of the system (e.g., parameters for tag merging, or the numerical values for trust levels as discussed in Section 4).

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). We would like to express our gratitude to Christoph Dorn, Daniel Schall, Florian Skopik, Josef Spillner, Jordan Janeiro and Markus Jung for help on various aspects of the paper.

References

[1] C. Schroth and T. Janner, "Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services," *IT Professional*, vol. 9, no. 3, pp. 36–41, 2007.

[2] 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.

[3] J. Soriano, D. Lizcano, J. J. Hierro, M. Reyes, C. Schroth, and T. Janner, "Enhancing User-Service Interaction through a Global User-Centric Approach to SOA," in *Proceedings of the Fourth International Conference on Networking and Services (ICNS)*, 2008.

[4] A. van Moorsel, "Metrics for the Internet Age: Quality of Experience and Quality of Business," HP Labs, Tech. Rep., 2001.

[5] A. Mathes, "Folksonomies – Cooperative Classification and Communication Through Shared Metadata." [Online]. Available: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

[6] E. Michlmayr, "A Case Study on Emergent Semantics in Communities," in *Proceedings of the Workshop on Social Network Analysis, International Semantic Web Conference (ISWC)*, 2005.

[7] A. Michlmayr, F. Rosenberg, C. Platzer, and S. Dustar, "Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective," in *Proceedings of the International Workshop on Service Oriented Software Engineering (IW-SOSE'07)*, 2007.

[8] L.-H. Vu, M. Hauswirth, and K. Aberer, "QoS-based Service Selection and Ranking with Trust and Reputation Management," in *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, 2005.

[9] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A QoS-Aware Selection Model for Semantic Web Services," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC)*, 2006.

[10] S. A. Golder and B. A. Huberman, "Usage Patterns of Collaborative Tagging Systems," *Journal of Information Science*, vol. 32, no. 2, pp. 198–208, 2006.

[11] S. Tai, N. Desai, and P. Mazzoleni, "Service Communities: Applications and Middleware," in *Proceedings of the 6th International Workshop on Software Engineering and Middleware (SEM)*, 2006.

[12] H. Meyer and M. Weske, "Light-Weight Semantic Service Annotations through Tagging," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC)*, 2006.

[13] I. Silva-Lepe, R. Subramanian, I. Rouvellou, T. Mikalsen, J. Diament, and A. Iyengar, "SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC)*, 2008.

[14] P. Resnick and H. R. Varian, "Recommender Systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[15] U. S. Manikrao and T. V. Prabhakar, "Dynamic Selection of Web Services with Recommendation System," in *Proceedings of the International Conference on Next Generation Web Services Practices (NWESP)*, 2005.

[16] A. Birukou, E. Blanzieri, P. Giorgini, and N. Kokash, "Improving Web Service Discovery with Usage Data," *IEEE Software*, vol. 24, no. 6, pp. 47–54, 2007.

[17] 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)*, 2007.

[18] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *Proceedings of the 17th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 421–430.

[19] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo," TUV-1841-2009-03, Vienna University of Technology, Tech. Rep., 2009.

[20] 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 the 12th IEEE International EDOC Conference*, 2008.

[21] C. Cattuto, D. Benz, A. Hotho, and G. Stumme, "Semantic Analysis of Tag Similarity Measures in Collaborative Tagging Systems," May 2008. [Online]. Available: http://arxiv.org/abs/0805.2045

[22] M. Baldauf, S. Dustdar, and F. Rosenberg, "A Survey on Context-Aware Systems," *International Journal of Ad Hoc Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[23] G. Palla, I. J. Farkas, P. Pollner, I. Derenyi, and T. Vicsek, "Fundamental statistical features and self-similar properties of tagged networks," *New Journal of Physics*, vol. 10, 2008.

[24] D. Schall, C. Dorn, S. Dustdar, and I. Dadduzio, "VieCAR – Enabling Self-adaptive Collaboration Services," in *Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2008.

[25] J. Spillner, M. Feldmann, I. Braun, T. Springer, and A. Schill, "Ad-Hoc Usage of Web Services with Dynvoker," in *Proceedings of the 1st European Conference Towards a Service-Based Internet (ServiceWave)*, 2008.

E Monitoring and Analyzing Influential Factors of Business Process Performance

Monitoring and Analyzing Influential Factors of Business Process Performance

Branimir Wetzstein* Philipp Leitner† Florian Rosenberg† Ivona Brandic† Schahram Dustdar† Frank Leymann*

*Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

†Distributed Systems Group
Vienna University of Technology
Vienna, Austria
lastname@infosys.tuwien.ac.at

***Abstract*—Business activity monitoring enables continuous observation of key performance indicators (KPIs). However, if things go wrong, a deeper analysis of process performance becomes necessary. Business analysts want to learn about the factors that influence the performance of business processes and most often contribute to the violation of KPI target values, and how they relate to each other. We provide a framework for performance monitoring and analysis of WS-BPEL processes, which consolidates process events and Quality of Service measurements. The framework uses machine learning techniques in order to construct tree structures, which represent the dependencies of a KPI on process and QoS metrics. These dependency trees allow business analysts to analyze how the process KPIs depend on lower-level process metrics and QoS characterisitics of the IT infrastructure. Deeper knowledge about the structure of dependencies can be gained by drill-down analysis of single factors of influence.**

I. Introduction

Business Process Management (BPM) encompasses a set of methods, techniques, and tools for modeling, executing and analyzing business processes of an organization [1]. Recently, BPM has been supported by a set of tools which have been integrated in order to support the business process lifecycle in a unified manner. Thereby, business analysts create a business process model, which is then refined by IT engineers to an executable model. The executable process model is deployed to a process engine, which executes the process by delegating tasks to humans and services. The execution of processes is often based on a Service Oriented Architecture [2] (SOA). In that case, the business process model is typically implemented as a service composition, for example in WS-BPEL [3].

An important aspect of the BPM lifecycle is the continuous supervision of business goals and timely measurement of business process performance. This is typically supported by business activity monitoring (BAM) technology, which enables continuous, near real-time monitoring of processes based on an eventing infrastructure [4]. Analysts define Key Performance Indicators (KPIs) and their target values based on business goals (e.g., "order fulfillment lead time $<$ 3 days"). KPIs are influenced by a set of Process Performance Metrics (PPM) [5], which are metrics based on process runtime data (e.g., "number of orders which can be served from inhouse stock"). PPMs are on a different level of granularity than KPIs: a KPI measures the success of the process as a whole, while a PPM captures only a single facet of the process, which is usually not interesting in isolation. Additionally, KPIs are influenced by technical parameters, i.e., the Quality of Service (QoS) metrics of the SOA (e.g., the availability of the process engine or the response time of Web services).

Business Activity Monitoring provides useful information on KPI achievement. However, the focus is set on the "what" rather than the "why" question. When KPIs do not meet target values the business analysts are interested in factors that cause these deviations. Since KPIs potentially depend on numerous lower-level PPMs and QoS metrics, these causes can be manifold, and are rarely obvious even to domain experts. In this paper we present an integrated framework for run-time monitoring and analysis of the performance of WS-BPEL processes. Our main contribution is the presentation of a framework for dependency analysis, a machine learning based analysis of PPMs and QoS metrics, with the ultimate goal of discovering the main factors of influence of process performance (i.e., KPI adherence). These factors are represented in an easy-to-interpret decision tree (dependency tree). We present the general concepts of our analysis framework, and provide experimental results based on a purchase order scenario, identify cases when dependency trees do not show expected results, and explain strategies how these problems can be coped with.

The rest of the paper is organized as follows. In Section II we present a scenario which we use for explaining our concepts and for experimentation and explain the research issues that this paper deals with in more detail. Section III explains the main ideas of our framework for runtime monitoring and dependency analysis. Section IV explains in detail the monitoring of influential factors, which is then followed by the description of the dependency analysis in Section V. Section VI describes the implementation of our prototype based on the scenario and experimental results, which are also extensively interpreted. Section VII discusses important related work and Section VIII finally concludes the paper and presents some future research directions.

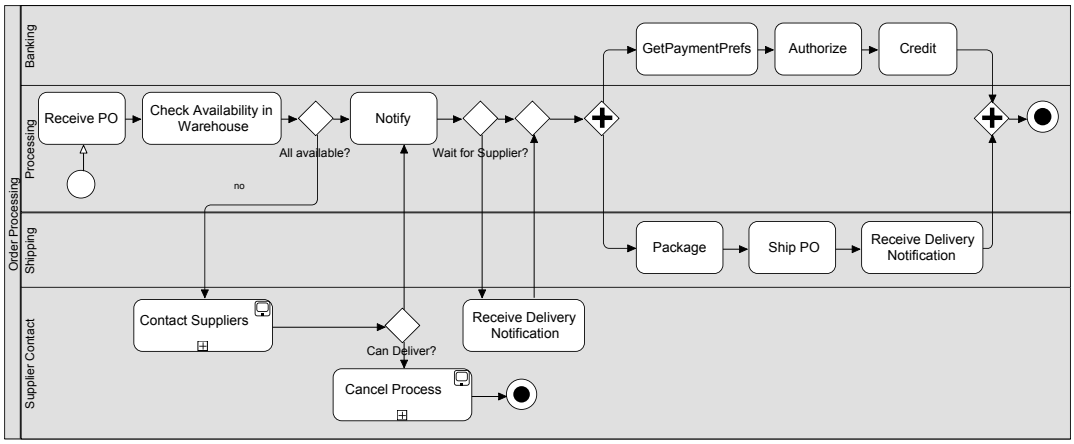


Fig. 1. Reseller Process Model in BPMN

II. Scenario

In this section we present a scenario which we use in the following sections for explaining our concepts and which we have implemented and used for experimentation purposes. We have chosen a purchase order scenario consisting of a customer, a reseller, its two suppliers, a banking service, and a shipping service. The business process of the reseller is illustrated in the BPMN diagram shown in Figure 1. The reseller offers certain products to its customers. It holds a certain part of the products in stock and orders missing products from suppliers if necessary. The customer sends a purchase order request with details about the required products and needed amounts to the reseller. The latter checks whether all products are available in stock. If some products are not in stock, they are ordered from suppliers. Note that the second supplier is contacted only if the first (preferred) supplier is not able to deliver. If the purchase order can be satisfied, the customer receives a confirmation, otherwise the order is rejected. The reseller waits, if needed, for the supplier to deliver the needed products. When all products are in place, the warehouse packages the products and hands them over to the shipment service, which delivers the order to the customer, and finally notifies the reseller about the shipment. In parallel to the packaging and shipment, the payment subprocess is performed. For that, the customer decides on the payment style and gives its payment details. The reseller contacts a banking service which authorizes the customer and credits the agreed amount. From the point of view of the reseller, a typical KPI is the *order fulfillment lead time* (duration from receiving the customer order until shipment is received by the customer), as defined in the Supply-Chain Reference Model (SCOR) [6].

Assuming that this process is implemented using WS-BPEL, the KPI *order fulfillment lead time* is potentially influenced by a number of technical and non-technical factors, such as the response time and availability of Web services, the customer, or the products ordered (Figure 2). In Table I, we have provided an (incomplete) list of potential factors of influence for the KPI from our scenario. Factors can include simple facts from the business process instance, such as a customer

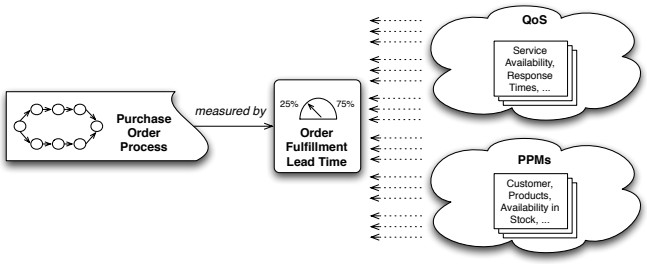


Fig. 2. KPIs, PPMs and QoS metrics

identifier, a product type, or information about which branch of a process has been executed (e.g., whether the alternative branch "ordering from external suppliers" needed to be executed). All these facts are accessible from the process instance, therefore, we have not given a calculation formula for these PPMs in the table. However, PPMs on a different level of granularity are also possible, such as the duration of a whole subprocess. Finally, we have given a few simple examples of QoS metrics, which may influence the KPI performance, such as the availability of the process engine or single services that the process relies on, or the response time of these services. A full discussion of possible QoS metrics is out of scope of this paper. The interested reader may refer e.g., to [7] for a more complete description of possible QoS metrics and their measurement. For completeness, we have also provided the possible range for these example influential factors. Generally, factors of influence can either be nominal values (i.e., take on one of a finite number of predefined values), or numeric values (e.g., integer or real values).

However, it is not obvious even to experts which of these factors actually influence the KPI most, and what the structure of the dependencies between factors of influence is (i.e., some factors are in turn influenced by others, such as the duration of the payment subprocess which is again influenced by service response times). These questions are not answered sufficiently by today's BAM dashboards – they can only provide status

Name	Type	Calculation Formula	Range
Customer ID	PPM		$\{Customer_1, Customer_2, \ldots\}$
Product Type	PPM		$\{Product_1, Product_2, \ldots\}$
Shipped from stock	PPM		$\{true, false\}$
Duration of Payment Subprocess	PPM	$t_{end} - t_{begin}$	$[0; \infty]$
Availability Process Engine	QoS	$\frac{\#available}{\#checks}$	$[0; 1]$
Availability Banking Service	QoS	$\frac{\#available}{\#checks}$	$[0; 1]$
Response Time Banking Service	QoS	$t_{end} - t_{begin}$	$[0; \infty]$

TABLE I
POTENTIAL INFLUENTIAL FACTORS OF KPI PERFORMANCE

information about KPIs, but do not allow further analysis of the main causes for violations. Our approach supports this kind of analysis, which we refer to as dependency analysis (i.e., the analysis of dependencies of KPIs to PPMs and QoS metrics). Furthermore, more detailed information about internal dependencies between factors of influence can be gained by drill-down analysis, i.e., recursively applying dependency analysis to single factors of influence.

III. FRAMEWORK OVERVIEW

In this section we describe the concepts of our framework for monitoring and analyzing factors of influence of business process performance. A high-level overview of the main components is given in Figure 3. In our framework we distinguish three different layers.

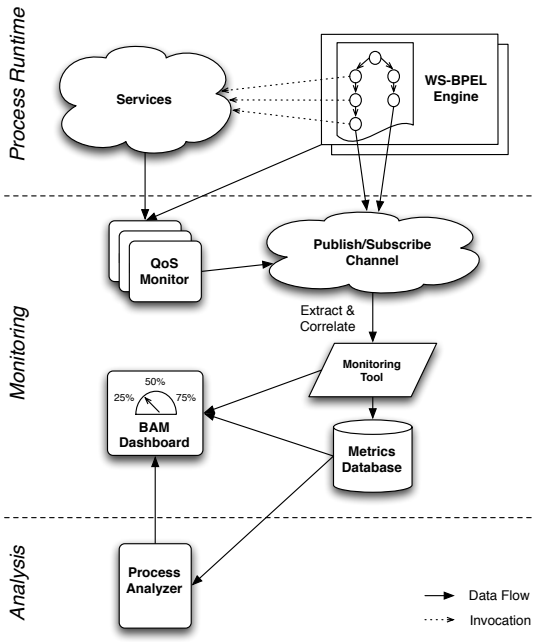


Fig. 3. Monitoring and Analysis Framework Overview

In the *process runtime* layer, a WS-BPEL business process is defined and executed. The process can be executed in a standard WS-BPEL compliant engine, as long as the engine is able to emit the process information necessary for calculating PPMs in form of process events.

In the *monitoring layer*, information about the running business process and the services it interacts with is collected in order to monitor KPIs, PPMs, and QoS metrics. Note that we assume that the user has defined a set of potential influential factors he wants to monitor for a KPI in order to ensure that corresponding metric data is available later on for analysis. KPIs and their potential infuential factors, consisting of both process performance metrics (PPMs) and QoS metrics, are modeled as part of a Process Metrics Definition Model (PMDM). Those metric definitions are deployed to the corresponding monitoring infrastructure components: QoS metric measurement directives to a QoS monitor(s); event definitions needed for PPM calculation to the WS-BPEL engine; and the whole PMDM to the monitoring tool. During process execution time, the QoS monitor and WS-BPEL process engine publish events to a publish/subscribe channel which the monitoring tool is subscribed to. The PPM and QoS metric values are calculated, stored in the metrics database for later analysis, and displayed in the BAM dashboard.

In the *process analysis layer*, the collected metrics information is analyzed by the process analyzer component. When the user is interested in performing a dependency analysis of KPIs, i.e., analyze the influential factors, the process analyzer gathers the needed metric data from the metrics database, prepares it for data mining, and uses a decision tree algorithm to generate a dependency tree which shows the influential factors of the KPI. Outcomes of the analysis are again displayed in the dashboard to the users of the system, who can use this resulting information to optimize the business process.

IV. MONITORING OF INFLUENTIAL FACTORS

We distinguish in our approach between PPMs and QoS metrics, which are supported by different monitoring mechanisms. PPMs are measured based on business events (e.g., OrderReceivedEvent) which are published by the WS-BPEL engine and other involved systems as the process instances are executed (Section IV-A). KPIs are specified over PPMs by providing a predicate (boolean-valued function) over their

values which evaluates to true if the target value is achieved, e.g., *order fulfillment lead time* < 2 days. QoS metrics are used for measuring the IT characteristics of the involved systems (e.g., availability and response time of Web service operations) and are provided either by dedicated QoS monitors or instrumentations of involved systems (Section IV-B).

A. PPM Monitoring

PPMs are metrics defined based on runtime events of processes. In the following, we focus on runtime events of WS-BPEL service orchestrations, but in general our approach supports arbitrary events of information systems participating in the business process.

PPM monitoring encompasses three phases: modeling, deployment, and monitoring. In the modeling phase, PPM definitions are specified in an XML file as part of the PMDM. A PPM definition specifies the name, description, unit, data type, and value calculation of the metric. When specifying PPMs we can distinguish between *atomic* and *composite* metrics. Atomic metrics are specified based on events which are published by the WS-BPEL engine. The metric value calculation of an atomic metric specifies how the value of the metric is retrieved from process event data (e.g., timestamp or a process data element). Composite metrics are calculated using diverse functions (arithmetic, aggregation, relational) based on atomic metrics and other composite metrics. When the PMDM is to be deployed to the monitoring tool, we generate *event filters* which subscribe to the events emitted by the process engine. Most engines support an event publishing mechanism, which in most cases is also to some extent configurable on which events to publish. During process execution, the PPM monitor receives events via event listeners which are subscribed to the publish/subscribe channel which the process engine uses for publishing events, and then calculates atomic metrics using event filters. After calculation of an atomic metric, its value is saved in the metrics database with a reference to the corresponding process instance. This reference is needed for later analysis. In the next step, all composite metrics which use that atomic metric are calculated in a recursive fashion.

In the following, we explain the PPM monitoring concepts based on a sample PPM from our scenario. Listing 4 shows a definition of an atomic metric in the PMDM which retrieves the information from a process variable on whether the supplier can deliver ordered products. To define an atomic PPM the user can use a set of predefined functions for measuring duration of activities and between activities, count the occurrence of activity executions, extracting the state of an activity or process, or access process data variables. The parameters of these predefined functions are linked to elements of process models such as a process activity, a data variable, or the process itself, and specify in addition the state of the process element when it is to be measured (e.g., started, halted, completed). In our example, we use a `variable` function which references an element containing that information in the corresponding process variable `orderItemsResponseMessage` and the activity `orderItemsFromSupplier1` at which the variable value

```
1  <ppm id="Supplier1_can_Deliver">
2   <name>Supplier1_can_Deliver</name
3   <dataType>boolean</dataType>
4   <calculation>
5    <calc:variable activity="orderItemsFromSupplier1"
6     variable="orderItemsResponseMessage" />
7   </calculation>
8   <attachments>
9    <activityAttachment
10    parameterName="orderItemsFromSupplier1"
11    xlink:type="simple"
12    xlink:href="PurchaseOrderProcess.bpel#xpointer(
13    //*[@name=&quot;orderItemsFromSupplier1&quot;])">
14    <activityStatus>completed</activityStatus>
15   </activityAttachment>
16   <variableAttachment
17    parameterName="orderItemsResponseMessage"
18    xlink:type="simple"
19    xlink:href="loanApprovalProcess.bpel#xpointer(
20    //*[@name=&quot;orderItemsResponseMessage&quot;])">
21    <variablePart>deliveryPossible</variablePart>
22   </variableAttachment>
23  </attachments>
24 </ppm>
```

Fig. 4. Sample PPM Definition

is to be read (lines 5-6). In the `attachments` block the links to corresponding WS-BPEL process elements are defined (lines 8-23). Thereby, we use XLink[1] and XPointer[2] to point to the XML elements in the WS-BPEL file. For example, we reference the activity with name `orderItemsFromSupplier1` in the `PurchaseOrderProcess.bpel` XML file (lines 12-13). In addition, we specify that at the state `completed` of that activity the corresponding event should be gathered (line 14). Note that we neglect here that activities can be performed several times per process instance if they are part of a loop; in that case one would have to additionally specify in which loop execution one is interested in.

When the PPM definition is to be deployed to the monitoring tool, assuming the usage of the Apache ODE BPEL Engine[3] (which we have also used for our scenario implementation), an event filter for the events `ActivityExecEndEvent` and `VariableModificationEvent` is generated. The first event is published when the corresponding activity has completed; it contains the process model name, the activity name, and identifiers of the corresponding process instance and activity instance. The second event is sent whenever a WS-BPEL variable has changed and contains the process variable data, the name of the process model, the name of the variable, and a process instance identifier.

The task of an event filter is to calculate an atomic metric as specified in the PPM definition based on received event data. Therefore, it has to deal with *event correlation* and *process instance management*. In the example above, for each process instance, the event filter collects (potentially more than one) `VariableModificationEvent` until it receives an `ActivityExecEndEvent`, and then chooses the *last* received `VariableModificationEvent` in order

[1] http://www.w3.org/TR/xlink/
[2] http://www.w3.org/TR/xptr-xpointer/
[3] http://ode.apache.org/

```
<qm id="PO_Process_Availability">
 <name>PO_Process_Availability</name
 <dataType>integer</dataType>
 <calculation>
  <availability>
   <endpoint>
     http://localhost:8082/.../poProcess?wsdl
   </endpoint>
   <testFrequencyPerMinute>20</testFrequencyPerMinute>
   <startTimePpm idref="PO_Process_Started_Time" />
   <endTimePpm idref="PO_Process_Completed_Time" />
  </availability>
 </calculation>
</qm>
```

Fig. 5. Sample QoS Metric Definition

to retrieve the most recent content of the needed process data variable. Thereby, those events are correlated using the process instance identifier which is part of both types of events. Note that in case of non-BPEL events the correlation could not be done based on the technical process identifiers (assigned by the BPEL engine), but based on a business identifier such as the `purchase order identifier`.

B. QoS Metrics Monitoring

The business processes we focus on in this paper are implemented as service compositions running on top of a SOA. Such processes have several dependencies on IT components and their QoS characteristics, which potentially influence business process performance. In our context there are three possibilities of measuring QoS: (1) probing by a separate QoS monitor, such as the one described in [7], (2) instrumentation of the WS-BPEL engine, or (3) instrumentation of the WS-BPEL process (evaluating QoS parameters using PPMs, e.g., response times of Web services can be estimated through WS-BPEL activity durations). In our scenario implementation, we use an external QoS monitor for measuring the availability of the process engine and partner Web services of the WS-BPEL process. The QoS monitor polls the corresponding endpoints and emits QoS events which contain information on their availability at a certain point in time. Response time is estimated based on the duration of the corresponding WS-BPEL `invoke` activity.

Just like PPMs, QoS metrics are defined in the PMDM. Listing 5 shows the definition of the availability metric for the purchase order process from our scenario. Thereby, we assume that availability is measured by an external QoS monitor by polling the corresponding Web service `endpoint` with a certain `testFrequencyPerMinute`. The QoS monitor will thus emit 20 QoS events per minute specifying whether the process was available. As the case for PPMs, an important aspect in QoS metric definition is their *correlation* with process instances. When a QoS metric is evaluated, it has to be assigned to process instances and/or activity instances it affects. In this context, there is a technical difference between an external QoS monitor and the instrumentation approaches considering the correlation of process instances and QoS events. As the instrumentation is internal to the engine, it has access to the context of the process instance. Thus, it is possible to write the process instance (and if needed activity instance) identifier as an attribute into the QoS event which can then be used to correlate the QoS event with the process instance. For example, in case of a response time measurement for an invoke activity, the engine instrumentation can include the process instance identifier and the activity instance identifier into the QoS measurement event. In case of an external QoS monitor this is not possible. Thus, in order to be able to correlate QoS measurements with the affected process instances for our sample metric, we specify in addition which QoS measurements (namely those between the start time `PO_Process_Started_Time` and completion time `PO_Process_Completed_Time` of the process instance) are to be taken into account when calculating the availability for a specific process instance.

V. Analysis of Influential Factors

The main idea of dependency analysis is to use historical process instances to determine the most important factors that dictate whether a process instance is going to violate its KPIs or not. The input of this analysis are stored metric values of process instances, which are available in the metrics database. The output of dependency analysis is a decision tree that incorporates the most important factors of influence of process performance. We refer to this tree as *dependency tree*, because it represents the main dependencies of the business process on technical and process metrics, i.e., the metrics which contribute "most often" to the failure or success of a process instance in respect to a KPI.

A. Background

In our approach we use decision tree learning, a well-established machine learning technique [8] for construction of dependency trees. Decision tree classifiers are a standard technique for supervised learning (i.e., concepts are learned from historical classifications, in our case dependency information is learned from monitoring of previous process instances). Decision trees follow a "divide and conquer" approach to learning concepts – they iteratively construct a tree of decision nodes, each consisting of a test; leaf nodes typically represent a classification to a category. In our case, only two categories exist (KPI has been violated, or not). One big advantage of decision tree algorithms in our context is their non-parametric nature: decision trees need only a very limited set of parameters (in the simplest case none) to work correctly, and can therefore be expected to provide useful results from the first run, without the need for extensive experiments with different parameter sets. Therefore, learning of dependency trees is completely automated and transparent to the user, allowing a business analyst to carry out dependency analysis with good results out of the box. We use the well-known C4.5 [9] and alternate decision tree (ADTree) [10] techniques, since their quality is well established in the community. For decision tree training we use 10-fold cross validation [8] to avoid having to split our historic process data into training, test and validation sets, and to estimate the classification error of the decision tree.

The classification error allows the business analyst to measure the quality of the dependency tree, i.e., how exact the tree represents the actual structure of real-world dependencies.

B. Creation of Dependency Trees

The inputs and outputs of dependency analysis are sketched in Figure 6. The analysis takes a set of historical process instance metric values as input (training set), and produces a tree representation of the internal dependencies.

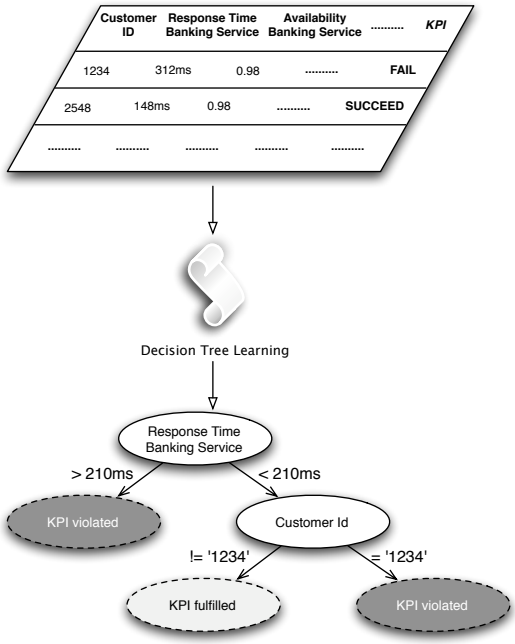


Fig. 6. Dependency Analysis Input and Output

The analysis consists of the following phases: (1) KPI selection and optional adjustment of analysis parameters, (2) creation of the training set, (3) decision tree learning, (4) displaying the result tree in the dashboard. All phases but the first one are performed automatically. In the first phase, the user chooses a KPI (from the PMDM) he wants to analyze. Optionally he can adjust the following parameters (or alternatively use default values): the *KPI target value* (and corresponding predicate); the *analysis period* and/or how many process instances in that period should be analyzed (e.g., last 1000); a subset of metric types from the PMDM which should be used as potential influential factors (default value: all); the decision tree algorithm which should be used. In the second phase, the creation of the training set is then performed automatically as follows: for each *process instance* which has begun and finished in the analysis period, the corresponding PPM and QoS metric values are gathered from the metrics database and used as attributes of a record of the training set. The predicate of the KPI metric value is evaluated and according to the result, the record is classified as "KPI fulfilled" or "KPI violated". An example training set is shown in Figure 6. Each row (record) contains the metric values (representing potential influential factors) of a process instance, whereby the last column specifies whether the KPI target value predicate (order fulfillment time $<$ target value) is fulfilled or violated for that process instance.

As a result of decision tree learning a dependency tree is constructed as shown in Figure 6 and displayed in the dashboard. In this (simple) example, the most influential factor is the response time of the banking service, since a delay in this service generally leads to a violated KPI. However, even if the banking services response time is acceptable (below 210 time units in this example), the KPIs are still often violated if the order is placed by the customer with the ID '1234'. Business analysts can use the dependency tree to learn about the "hot spots" of the process, and inform themselves about possible corrective actions if a process underperforms. For example, considering the example in Figure 6, a business analyst can take the corrective action to replace the "Banking Service" against a service with better response time, if such a service is available. However, note that the "first" metric used in the dependency tree is not necessarily the most important one – to find out about the most important metrics one needs to look at the whole tree and find out which decisions lead to the most failed process instances. For metrics which have been identified as factors of influence, a further "drill down" analysis can be performed. For this, one of the factors of influence (e.g., the response time of the banking service) is selected, a target value is specified, and another dependency analysis is launched. This identifies the more detailed dependencies that influence this specific factor of the overall process performance (e.g., one could find out that way that the response time of the banking service strongly depends on the type of the banking account).

VI. Experimentation

In this section we describe our prototype implementation of the framework and experimental results based on the example scenario.

A. Experiment Setup and Implementation

We have implemented the scenario as presented in Section II using a Java-based prototype. We use Apache ODE as our business process execution engine. ODE is open source software, and implements the WS-BPEL standard for Web service orchestration. One of the features of ODE is that it can trigger execution events (e.g., `ActivityExecStartEvent`), which we use as process events as described in Section III. The eventing features of ODE demand for a JMS implementation to take care of the transportation of events to subscribers (the monitoring tool in our case). We chose to use the open source message queue Apache ActiveMQ[4], however, any other JMS implementation could be used as well. The purchase order process has been implemented as a WS-BPEL process which interacts with six Web services including the client of the process. These Web services have been implemented in Java using Apache CXF[5] and simulate certain influential

[4]http://activemq.apache.org/
[5]http://cxf.apache.org/

factors. One can, for example, configure the response time, availability, and outputs of a service over time and dependant on business process data. The metrics database is implemented as a standard MySQL[6] database. Because of the limited size of the scenario we did not use advanced features such as clustering or load balancing. The Monitoring Tool for evaluation of PPMs has been implemented in Java as described in [5]. We have additionally implemented support for correlation of PPMs and QoS metrics. The Dashboard component is implemented as an standalone Swing application. The process analyzer is a standalone Web service, which is accessible over a RESTful interface. The foundation of this component is the WEKA toolkit[7], which implements many high-quality machine learning schemes, including the decision tree based classifiers that we used in this paper. We have transparently integrated WEKA into our process analyzer component using the WEKA Java API. Finally, we have implemented a simple QoS monitor, which can non-intrusively check the availability of Web services through periodic polling. This QoS data is again provided to the monitoring dashboard through a RESTful interface. For experimentation, we have deployed all these components on a single desktop PC, mainly to prevent external influences such as network latency to influence our experimentation results. However, the scenario is designed in such a way that physically distributed experiments can be run without any modifications.

B. Experimental Results

The procedure of experimentation is as follows. We create a configuration which simulates certain influential factors and define a set of potential influential metrics. We then execute the process a certain number of times (100, 400, and 1000 times) by triggering the process using a simulated client. During execution, the process is monitored and metrics are saved in the metrics database. We then perform dependency analysis of the KPI and compare the result of the generated dependency tree with our configured influential metrics. In the following we present the results of two experimental runs. For both of them, we have used the same configuration consisting of the KPI *Order Fulfillment Lead Time* and a set of 31 potential influential factors (a subset is shown in Table I).

For the first run, we have created a configuration which simulates the following factors: (i) the warehouse availability check (*order in stock*) returns a negative result for certain *product types* based on certain probabilities; *order in stock* is an important influential factor of the overall duration of the process as it decides whether products have to be requested from suppliers which increases the overall process duration substantially (ii) supplier 1 has in average a higher than expected *supplier delivery time*; (iii) average *shipment delivery time* is high in relation to the overall duration of the process instance. Based on this configuration, we expect the KPI to be mainly influenced by *order in stock*, *product type*, *supplier 1 delivery time*, and *shipment delivery time*. Other metrics (in particular response times of services) also influence the KPI value, but in a marginal way.

The generated decision tree is shown in Figure 7a. It has been generated using J48 (the WEKA implementation of C4.5 [9]) based on 100 process instances. The most influential factor is the *shipment delivery time*; if it is above 95 time units all process instances lead to KPI violations ("red"), otherwise they depend further on the *order in stock* metric and *supplier 1 delivery time*. The leaves of the tree show the number of instances which are classified as "red" or "green".

The dependency tree shows three of the four influential factors we have configured. Interestingly, the fourth factor, the *product type*, is not shown. The reason for this is that *product type* directly influences *order in stock*, which again influences the KPI value which is shown in the tree; as both metrics influence the KPI value in the same way, only one of them is shown in the tree. This particular result is unsatisfactory, as it hides the root cause, namely *product type* in this case. The user can deal with this problem using two approaches: (i) he can drill down and request the analysis of the *order in stock* metric. A second tree is generated which explains when ordered products are not in stock as shown in Figure 7b. This tree now clearly shows how the unavailability depends on *product type* and *ordered product quantity*. (ii) The user can also remove the *order in stock* metric temporarily from the analyzed metric set. Now, the algorithm will search for alternative metrics which classify the instances in a similar way as *order in stock*. In that case, as shown in the experiments, the algorithm finds and displays *product type* in the tree (not shown in the figure).

Table II shows the more detailed results of the first experimental run. We have experimented with two algorithms: J48 (based on C4.5 [9]) and ADTree (alternating decision tree [10]). Both of them show very similar results concerning the displayed influential metrics. Typically there is only one or at most two (marginal) metrics which differ. For the same precision (correctly classified instances in the training set, as shown in the last column), the algorithms also generate trees of about the same size. The usage of parameters has lead to only marginal changes in our experiments (for example, J48 -U with no pruning). The only parameter that turned out useful in our experimentation was the "reduced error pruning" (J48 -R) [8] as it reduced the size of the tree, loosing accuracy only marginally. This parameter is useful as the experiments show that the tree is getting bigger (column "Leaves/Nodes") with the number of process instances. For example, J48 generated for 400 instances a tree with 11 nodes, for 1000 instances a tree with 18 nodes, while the precision improved only by 1%. In particular, when the tree gets bigger, factors are shown in the tree which have only marginal influence and thus make the tree less readable; column "Displayed Metrics" shows how many distinct metrics are displayed in the tree, the first number thereby depiciting the number of expected metrics. In the case of too many undesirable (marginal) metrics, one can try to improve the result by simply removing those metrics from the analyzed metric set and repeating the analysis. Finally,

[6]http://www.mysql.com/
[7]http://www.cs.waikato.ac.nz/ml/weka/

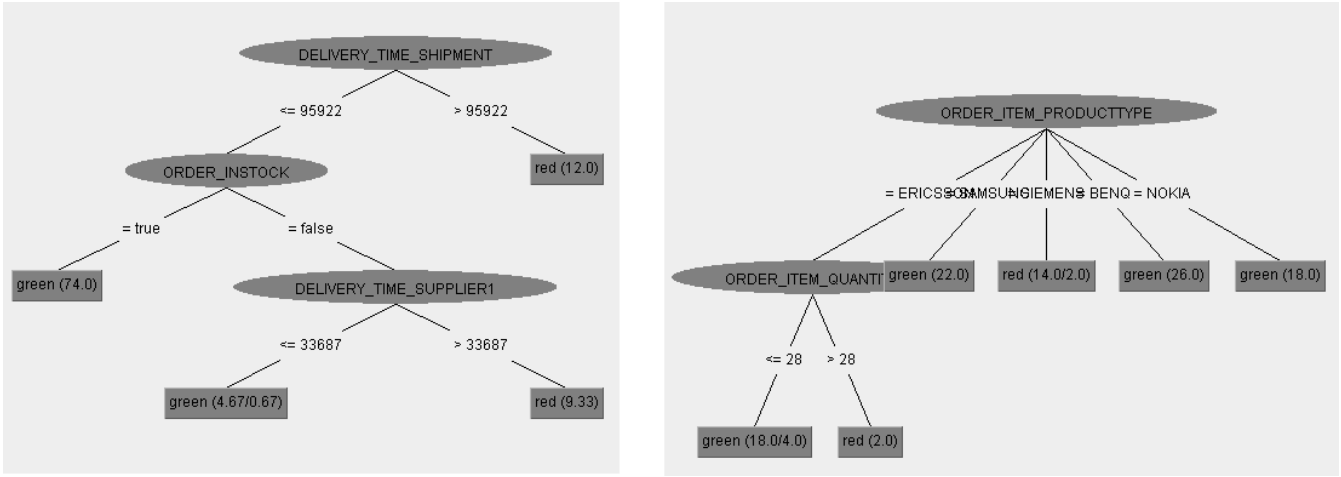


Fig. 7. Generated Trees for (a) *Order Fulfillment Time*, (b) *Order in Stock*

concerning the analysis duration, in our setting on a standard laptop computer a decision tree generation based on 1000 instances takes about 30 seconds.

Instances	Algorithm	Leaves/ Nodes	Displayed Metrics Expected/All	Correctly Classfied
100	J48	4/7	3/4	95,0 %
100	ADTree	11/16	2/4	98,0 %
400	J48	6/11	3/4	97,8 %
400	ADTree	17/26	4/5	99,0 %
1000	J48	11/18	3/6	98,8 %
1000	J48 -R	6/11	3/4	97,9 %
1000	J48 -U	13/22	4/9	99,2 %
1000	ADTree	19/28	3/6	99,4 %

TABLE II
EXPERIMENTAL RESULTS

For the second run, we have created a configuration which, among others, simulates QoS influential factors: (i) the warehouse Web service and the shipment Web service are unavailable with the probability of 15%; the BPEL process contains fault handlers when trying to invoke partner services; in case of unavailability it waits for a certain time frame and retries; we have defined *response time* metrics (measured based on process events) for each invoke-activity which "include" the retries in case of unavailability (ii) the warehouse availability check (*order in stock*) returns now a negative result only with the probability of 5%; (iii) *shipment delivery time* is still very influential in relation to the duration of other activities of the process. Based on this configuration, we expect the KPI to be mainly influenced by the *availability* of warehouse Web service and shipment Web service, *order in stock*, and *shipment delivery time*.

The generated decision tree is shown in Figure 8a. It is a J48 tree based on 1000 instances using reduced error pruning. The tree shows the *response time warehouse*, *delivery time shipment*, and *order in stock* as the main influential factors. Completely missing, however, are the expected dependencies on the availability of the warehouse Web service and the shipment Web service. We suspect that *availability of warehouse Web service* is hidden by *response time warehouse*, which could be analyzed by drilling down. However, we take now another approach and perform an analysis of the KPI only in relation to availability metrics of all services involved in the process, i.e., we remove all other metrics from the analyzed metric set. Effectively, we analyze the impact of availability on the KPI. The result is shown in Figure 8b. It clearly shows that (only) availability of the shipment and warehouse Web services have an impact on the KPI value, as expected.

Overall, we can draw the following conclusions from the experiments. In general, the generated trees show the expected influential metrics in a satisfactory manner. As expected, the non-parametric nature of decision tree algorithms makes dependency analysis produce suitable results "out of the box". We argue that this renders our approach suitable for non-IT personnel. However, this claim has yet to be verified through real-life evaluation. Concerning the influential factors displayed in the tree, we have identified two problems: (i) as the tree gets bigger it contains often more metrics than expected, i.e. metrics which have only marginal influence and thus only "blur the picture"; in that case one can try to tune the algorithm by using, for example, reduced error pruning, or one can simply remove those metrics from the analyzed metric set and repeat the analysis; both techniques lead to more satisfactory results; (ii) the tree does not show some of the expected metrics: we have shown that this is often the case when there are "multi-level" dependencies between metrics; in that case further analysis (drill down) of lower-level metrics may help to find further influential factors. While in the general analysis case, the user does not need to have any special domain knowledge on metric dependencies, in the drilldown case, we assume that the user *suspects* that there could be further dependencies behind a

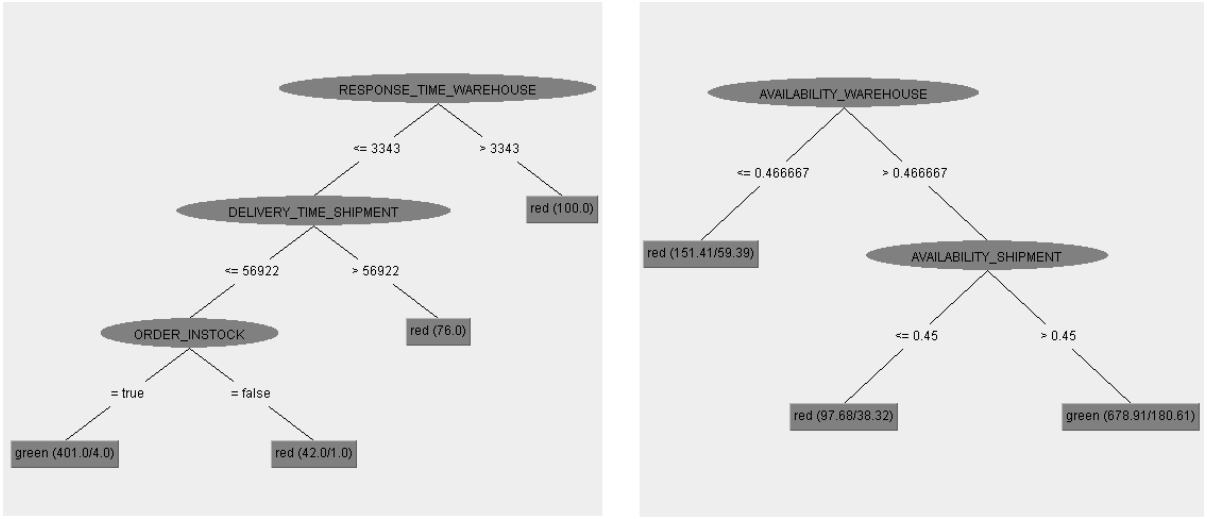


Fig. 8. Generated Trees for (a) All Factors, (b) Availability of IT Infrastructure

lower-level metric.

VII. Related Work

There are several approaches that deal with monitoring of service compositions. They differ mostly in monitoring goals, i.e., what is monitored, and the monitoring mechanisms. IBM's approach integrates performance management tightly into the business process lifecycle and supports it through its WebSphere family of products [11]. Thereby, process metrics are modeled and monitored based on events published by the Process Server BPEL engine. Our approach is similar to IBM's approach in that we use process events published by the process engine. We, however, also support monitoring of QoS metrics and automated dependency analysis. Baresi et al. [12] deal with monitoring of WS-BPEL processes focusing on runtime validation. The goal is thereby not to monitor process performance metrics, but to detect partner services which deliver unexpected results concerning functional expectations. Traverso et al. [13] describe a monitoring approach for WS-BPEL processes which supports run-time checking of assumptions under which the partner services are supposed to participate in the process and the conditions that the process is expected to satisfy. The approach supports also collecting statistical and timing information. All of these approaches have in common that they concentrate only on monitoring of business processes. In particular, they do not deal with QoS metrics integration and dependency analysis.

When it comes to the analysis aspect of service compositions, the idea of using dependency models for representing dependencies among different impact factors is not new and has been applied in connection with the monitoring of SLAs of service compositions in [14]. This approach focuses on dependencies of SLAs of overall service compositions on SLAs of composed services and analyzes reasons for SLA violations. Focusing on response time and cost metrics, the dependency relations and the impacts factors are identified at design time, and then later compared with monitoring results during runtime. In our approach, we do not model dependencies at design time, but solely construct the dependency model based on monitoring results using data mining. Most closely related to our work is the platform for business operation management developed by HP [15], as it supports both process monitoring, and analysis and prediction based on data mining. In [16] the authors give an overview and a classification of which data mining techniques are suitable for which analysis and prediction techniques. Thereby, also decision trees are mentioned as one possible technique, which is what we concentrate on in our approach. The platform presented allows users to define and monitor business metrics (not focused on WS-BPEL processes), perform intelligent analysis on them to understand causes of undesired metric values, and predict future values. Our approach is different in that we focus on SOA-based WS-BPEL processes, and explicitly integrate PPMs and QoS metrics for analysis purposes. We deal only with decision trees, but provide detailed experimental results. Another popular approach to process analysis is process mining. Process mining techniques operate on event logs provided by information systems and perform different kinds of analysis on them, in particular process discovery when there is no explicit process model a priori [17]. In our approach, we also operate on monitored "metric logs" during analysis phase, but focus on mining of metric dependencies using decision tree algorithms.

VIII. Conclusions

In this paper we have presented a framework that performs monitoring of both PPMs and QoS metrics of business processes runnning on top of a Service-Oriented Architecture. Besides providing up-to-date dashboard information about the current process performance, the main goal of our framework is to enable what we refer to as dependency analysis, i.e., an analysis of the main factors that influence the business process and make it violate its performance targets. The result of this analysis is represented as a decision tree. We have

presented experimental results which show that in general the generated decision trees provide explanations in a satisfactory manner, but in some cases further analysis has to be done. In that respect, we have shown how drill-down functionality and analyis based on different metric sets can influence the analysis result. One important advantage of our approach is that results are satisfactory without the need for extensive experimentation and parameter adaptation, which lets our approach seem feasible even for domain experts which are not IT-savy.

Our future work includes extending the framework presented here into various directions. Firstly, we plan to work on the runtime prediction of the outcome of process instances (i.e., whether the KPI is going to be violated or not) while they are still running. Basically, we can use the same techniques as for dependency analysis (however, we will in addition use regression trees). Secondly, we are working towards making use of the dependency analysis in the area of process adaptation – currently, dependencies are presented towards the human business analyst, who is then incorporating the gained knowledge back into the process, e.g., by exchanging service bindings. We currently think about a more automated feedback mechanism, which uses rule sets and predefined reactions to incorporate dependency knowledge back into the WS-BPEL process in a more automated way. One example would be service selection: if the dependency model of a process shows that the process outcome is sensitive to the response time of a service, then an expensive high-quality service is selected; if the response time is no important factor of influence a cheaper service is selected. Finally, we still need to test our approach in real-world settings, to further validate the claims that we have stated in this paper.

ACKNOWLEDGMENT

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

REFERENCES

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 11, 2007.

[3] OASIS, *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007.

[4] J.-J. Jeng, J. Schiefer, and H. Chang, "An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises," in *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC '03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 86.

[5] B. Wetzstein, S. Strauch, and F. Leymann, "Measuring Performance Metrics of WS-BPEL Service Compositions," in *The Fifth International Conference on Networking and Services (ICNS 2009), Valencia, Spain, April 20-25, 2009*. IEEE Computer Society, April 2009.

[6] S. Council, "Supply Chain Operations Reference Model Version 7.0," 2005.

[7] 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.

[8] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, 1993.

[10] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," in *Proceedings of the 16th International Conference on Machine Learning (ICML '99)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 124–133.

[11] U. Wahli, V. Avula, H. Macleod, M. Saeed, and A. Vinther., *Business Process Management: Modeling Through Monitoring Using WebSphere V6.0.2 Products*. IBM, International Technical Support Organization, 2007.

[12] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the 3rd International Conference of Service-Oriented Computing (ICSOC'05)*. Springer, 2005, pp. 269–282.

[13] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in *Proceedings of the IEEE International Conference on Web Services(ICWS'06)*, 2006, pp. 63–71.

[14] L. Bodenstaff, A. Wombacher, M. Reichert, and M. C. Jaeger, "Monitoring Dependencies for SLAs: The MoDe4SLA Approach," in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC '08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 21–29.

[15] M. Castellanos, F. Casati, M.-C. Shan, and U. Dayal, "iBOM: A Platform for Intelligent Business Operation Management," in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, 2005, pp. 1084–1095.

[16] M. Castellanos, F. Casati, U. Dayal, and M.-C. Shan, "A Comprehensive and Automated Approach to Intelligent Business Processes Execution Analysis," *Distributed and Parallel Databases*, vol. 16, no. 3, pp. 239–273, 2004.

[17] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

F Runtime Prediction of Service Level Agreement Violations for Composite Services

Runtime Prediction of Service Level Agreement Violations for Composite Services

Philipp Leitner[1], Branimir Wetzstein[2], Florian Rosenberg[3], Anton Michlmayr[1], Schahram Dustdar[1], Frank Leymann[2]

[1] Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040, Vienna, Austria
lastname@infosys.tuwien.ac.at

[2] Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

[3] CSIRO ICT Centre
GPO Box 664
Canberra ACT 2601, Australia
florian.rosenberg@csiro.au

Abstract. SLAs are contractually binding agreements between service providers and consumers, mandating concrete numerical target values which the service needs to achieve. For service providers, it is essential to prevent SLA violations as much as possible to enhance customer satisfaction and avoid penalty payments. Therefore, it is desirable for providers to predict possible violations before they happen, while it is still possible to set counteractive measures. We propose an approach for predicting SLA violations at runtime, which uses measured and estimated facts (instance data of the composition or QoS of used services) as input for a prediction model. The prediction model is based on machine learning regression techniques, and trained using historical process instances. We present the architecture of our approach and a prototype implementation, and validate our ideas based on an illustrative example.

1 Introduction

In service-oriented computing [1], finer-grained basic functionality provided using Web services can be composed to more coarse-grained services. This model is often used by Software-as-as-Service providers to implement value-added applications, which are built upon existing internal and external Web services. Very important for providers and consumers of such services are Service Level Agreements (SLAs), which are legally binding agreements governing the quality that the composite service is expected to provide (Quality of Service, QoS) [2]). SLAs contain Service Level Objectives (SLOs), which are concrete numerical

target values (e.g., "maximum response time is 45 seconds"). For the provider it is essential to not violate these SLOs, since typically violations are coupled with penalty payments. Additionally, violations can negatively impact service consumer satisfaction. Therefore, it is vitally important for the service provider to be aware of SLA violations, in order to react to them accordingly.

Typically, SLA monitoring is done *ex post*, i.e., violated SLOs can only be identified after the violation happened. While this approach is useful in that it alerts the provider to potential quality problems, it clearly cannot directly help preventing them. In that regard an *ex ante* approach is preferable, which allows to predict possible SLA violations before they have actually occurred. The main contribution of this paper is the introduction of a general approach to prediction of SLA violations for composite services, taking into account both QoS and process instance data, and using estimates to approximate not yet available data. Additionally, we present a prototype implementation of the system and an evaluation based on an order processing example. The ideas presented here are most applicable for long-running processes, where human intervention into problematic instances is possible. Our system introduces the notions of checkpoints (points in the execution of the composition where prediction can be done), facts (data which is already known in a checkpoint, such as the response times of already used services) and estimates (data which is not yet available, but can be estimated). Facts and estimates can refer to both typical QoS data (e.g., response times, availability, system load) and process instance data (e.g., customer identifiers, ordered products). Our implementation uses regression classifiers, a technique from the area of machine learning [3], to predict concrete SLO values.

The rest of the paper is structured as follows. In Section 2 we briefly introduce an illustrative example which we will use in the remainder of the paper. In Section 3 we detail the general concepts of our prediction approach. In Section 4 we described the implementation of a prototype tool, which we use for evaluation in Section 5. Finally, we provide an overview of relevant related work in Section 6 and conclude the paper in Section 7.

2 Illustrative Example

To illustrate the ideas presented in this paper we will use a simple purchase order scenario (see Figure 2 below). In this example there are a number of roles to consider: a reseller, who is the owner of the composite service, a customer, who is using it, a banking service, a shipping service, and two external suppliers. The business logic of the reseller service is defined as follows. Whenever the reseller service receives an order from the customer, it first checks if all ordered items are available in the internal stock. If this is not the case, it checks if the missing item(s) can be ordered from Supplier 1, and, if this is not the case, from Supplier 2. If both cannot deliver the order has to be cancelled, otherwise the missing items are ordered from the respective supplier. When all ordered items are available she will (in parallel) proceed to charge the customer using the banking service and initialize shipment of the ordered goods (using the Shipping

Service). We have borrowed this example from [4], please refer to this work for more information.

In this case study, the reseller has an SLA with its customers, with an SLO specifying that the end-to-end response time of the composition cannot be more than a certain threshold of time units. For every time the SLO is violated the customer is contractually entitled a discount for the order. Note that even though our explanations in this paper will be based on just one single SLO, our approach can be generalized to multiple SLOs. Additionally, even though we present our approach based on a numerical SLO, our ideas can be also applied to estimation of nominal objectives. However, SLOs need to adhere to the following requirements: (1) they need to be non-deterministic (following the definition in [5]), and (2) they cannot be defined as aggregations over multiple executions. Requirement (1) is not so much functionally important, but our prediction approach is not very useful otherwise (e.g., for SLOs concerning security requirements). Requirement (2) is a limitation of our current approach, which we plan on working on as part of our future work.

3 Predicting SLA Violations

In this section we present the core ideas of our approach towards prediction of SLA violations. Generally, the approach is based on the idea of predicting concrete SLO values based on whatever information is already available at a concrete point in the execution of a composite service. We distinguish three different types of information. (1) **Facts** represent data which is already known at prediction time. Typical examples of facts are the QoS of already used services, such as the response time of a service which has already been invoked in this execution, or instance data which has either been passed as input or which has been generated earlier in the process execution. (2) **Unknowns** are the opposites of facts, in that they represent data which is entirely unknown at prediction time. Oftentimes, instance data which has not yet been produced falls into this category. If important factors are unknown at prediction time the prediction quality will be very bad, e.g., in our illustrative example a prediction cannot be accurate before it is known whether the order can be delivered from the reseller's internal stock. (3) **Estimates** are a kind of middle ground between facts and unknowns, in that they represent data which is not yet available, but can be estimated. This is often the case for QoS data, since techniques such as QoS monitoring [5] can be used to get an idea of e.g., the response time of a service before it is actually invoked. Estimating instance data is more difficult, and generally domain-specific.

The overall architecture of our system is depicted in Figure 1. The most important concept used is that the user defines **checkpoints** in the service composition, which indicate points in the execution where a prediction should be carried out. The exact point in the execution model which triggers the checkpoint is called the **hook**. Every checkpoint is associated with one **checkpoint predictor**. Essentially, the predictor uses a function taking as input all facts

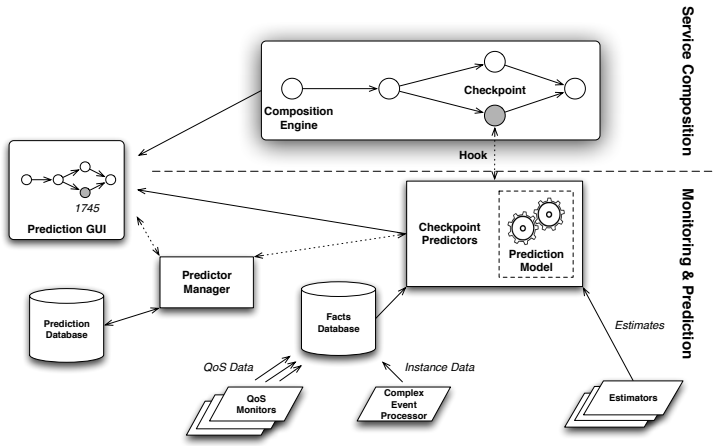


Fig. 1: Overall System Architecture

which are already available in the checkpoint, and, if applicable, a number of estimates of not yet known facts, and produces a numerical estimation of the SLO value(s). This function is generated using machine learning techniques We refer to this function as the **prediction model** of a checkpoint predictor. Facts are retrieved from a **facts database**, which is filled using a number of **QoS monitors** (which provide QoS data) and a **Complex Event Processing** (CEP) engine (which extracts and correlates the instance data, as emitted by the process engine). A detailed discussion of our event-based approach to monitoring is out of scope of this paper, but can be reviewed in related work [4, 6]. **Estimators** are a generic framework for components which deliver estimates. Finally, the prediction result is transferred to a graphical user interface (**prediction GUI**), which visualizes the predicted value(s) for the checkpoint. A **predictor manager** component is responsible for the lifecycle management of predictors, i.e., for initializing, destroying and retraining them. Additionally, predictions are stored in a **prediction database** to be available for future analysis.

3.1 Checkpoint Definition

At design-time, the main issue is the definition of checkpoints in the composition model. For every checkpoint, the following input needs to be provided: (1) The hook, which defines the concrete point in the execution that triggers the prediction, (2) a list of available facts, (3) a list of estimates, and the estimator component as well as the parameters used to retrieve or calculate them, (4) the retraining strategy, which governs at which times a rebuilding of the prediction model should happen, and (5) as a last optional step, a parameterization of the machine learning technique used to build the prediction model. After all these

inputs are defined the checkpoint is deployed using the predictor manager, and an initial model is built. For this a set of historical executions of the composite service need to be available, for which all facts (including those associated with estimates) have been monitored. If no or too little historical data is available the checkpoint is suspended by the predictor manager until enough training data has been collected. The amount of data necessary is case-specific, since it vastly depends on the complexity of the composition. We generally use the Training Data Correlation as a metric for evaluating the quality of a freshly trained model (see below for a definition), however, a detailed discussion of this is out of scope of this paper. After the initial model is built the continuous optimization of the predictor is governed by the predictor manager, according to the retraining strategy. Finally, the checkpoint can be terminated by the user via the prediction GUI. We will now discuss these concepts in more depth.

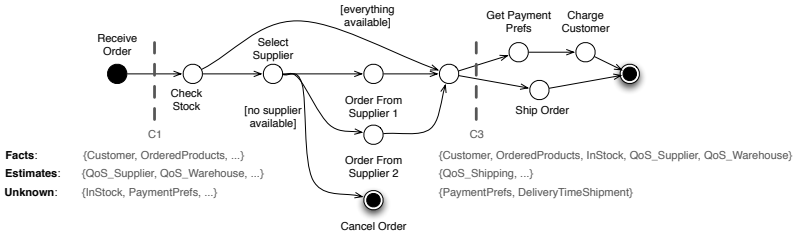


Fig. 2: Illustrative Example With Possible Checkpoints

Hooks Hooks can be inserted either before or after any WS-BPEL activity (for instance, an *Invoke* activity). Generally, there is a tradeoff to take into account here, since early predictions are usually more helpful (in that they rather allow for corrections if violations are predicted), but also less accurate since less facts are available and more estimates are necessary. Figure 2 depicts the (simplified) example from Section 2, and shows two possible checkpoints. In C_1 the only facts available are the ones given as input to the composition (such as a customer identifier, or the ordered products). Some other facts (mainly QoS metrics) can already be estimated, however, other important information, such as whether the order can be served directly from stock, is simply unavailable in C_1, not even as an estimate. Therefore, the prediction cannot be very accurate. In checkpoint C_3, on the other hand, most of the processes important raw data is already available as facts, allowing for good predictions. However, compared to C_1, the possibilities to react to problems are limited, since only the payment and shipping steps are left to adapt (e.g., a user may still decide to use express shipping instead of the regular one if a SLA violation is predicted in C3). Finding good checkpoints at which the prediction is reasonably accurate and still timely enough to react to problems demands for some domain knowledge about influential factors of composition performance. Dependency analysis as discussed in [6]

can help providing this crucial information. Dependency analysis is the process of using historical business process instance data to find out about the main factors which dictate the performance of a process. When defining checkpoints, a user can assume that the majority of important factors of influence need to be available as either facts or at least as good estimates in order to achieve accurate predictions.

Facts and Estimates: Facts represent all important information which can already be measured in this checkpoint. This includes both QoS and instance data. Note that the relationship between facts and the final SLO values does not need to be known (e.g., a user can include instance data such as user identifiers or ordered items, even if she is not sure if this has any relevance for the SLO). However, dependency analysis can again be used to identify the most important facts for a checkpoint. Additionally, the user can also define estimates. In the example above, in C_1 the response time of the warehouse service is not yet known, however, it can e.g., be estimated using a QoS monitor. Since estimating instance data is inherently domain-specific, our system is extensible in that more specific estimators (which are implemented as simple Java classes) can be integrated seamlessly. Estimates are linked to facts, in the sense that they have to represent an estimation of a fact which will be monitorable at a later point.

Retraining Strategy: Generally, the prediction model needs to be rebuilt whenever enough new information is available to significantly improve the model. The retraining strategy is used to define when the system should check whether rebuilding the prediction model is necessary. Table 1 summarizes all retraining strategies available, and gives examples. The custom strategy is defined using Java code, all other strategies are implemented in our prototype and can be used and configured without any additional code.

Strategy	**Retrains . . .**	**Example**
`periodic`	. . . in fixed intervals	*every 24 hours*
`instance-based`	. . . whenever a fixed number of new instances have been received since the last training	*every 250 instances*
`on demand`	. . . on user demand	–
`on error`	. . . if the mean prediction error exceeds a given threshold	*if* $\bar{e} > T$
`custom`	. . . if a user-defined condition applies	*whenever more than 10 orders from customer 12345 have been received*

Table 1: Predictor Retraining Strategies

Prediction Model Parameterization: A user can also define the machine learning technique that should be used to build the prediction model. This is done

by specifying an algorithm and the respective parameterization for the WEKA toolkit[4], an open source machine learning toolkit which we internally use in our prototype implementation. In this way the prediction quality can be tuned by a machine learning savvy user, however, we also provide a default configuration which can be used out of the box.

3.2 Run-Time Prediction

At runtime, the prediction process is triggered by lifecycle events from the WS-BPEL engine. These are events emitted by some engines (such as Apache ODE[5]), which contain lifecycle information about the service composition (e.g., `ActivityExecStartEvent`, `VariableModificationEvent`, `ProcessCompletionEvent`). Our approach is based on these events, therefore, a WS-BPEL engine which is capable of emitting these events is a preliminary of our approach. When checkpoints are deployed we use the hook information to register respective event listeners. For instance, for a checkpoint with the hook "After invoke CheckStock" we generate a listener for `ActivityExecEndEvent`s which consider the invoke activity "CheckStock". We show the sequence of actions which is triggered as soon as such an event is received in Figure 3.

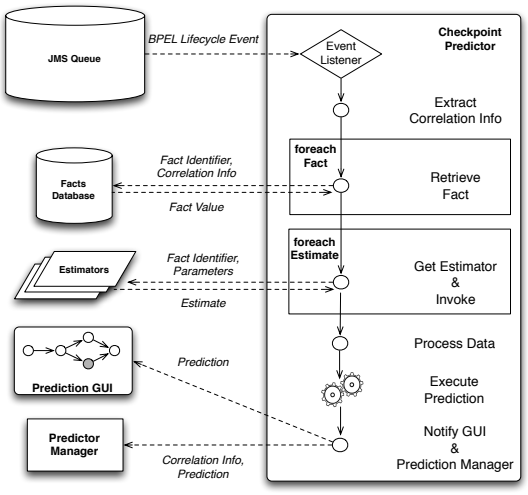


Fig. 3: Runtime View On Checkpoint Predictors

After being triggered by a lifecycle event the checkpoint predictor first extracts some necessary correlation information from the event received. This in-

[4] `http://www.cs.waikato.ac.nz/ml/weka/`
[5] `http://ode.apache.org`

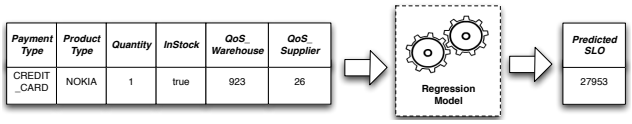


Fig. 4: Black-Box Prediction Model

cludes the process instance ID as assigned by the composition engine, the instance start time (i.e., the time when the instance was created) and the timestamp of the event. This information is necessary to be able to retrieve the correct facts from the facts database, which is done for every fact in the next step (e.g., in order to find the correct fact "CustomerNumber" for the current execution the process instance ID needs to be known). When all facts have been gathered, the predictor also collects the still missing estimates. For this, for every estimate the predictor instantiates the respective estimator component (if no instance of this estimator was available before), and invokes it (passing all necessary parameters as specified in the checkpoint definition). The gathered facts and estimates are then converted into the format expected by the prediction model (in the case of our prototype, this is the WEKA Attribute-Relation File Format ARFF[6]), and, if necessary, some data cleaning is done. Afterwards, the actual prediction is carried out by passing the gathered input to the prediction model producing a numerical estimation of the SLO value. This prediction is then passed to the prediction GUI (for visualization) and the prediction manager.

Note that the "intelligence" that actually implements the prediction of the SLO values is encapsulated in the prediction model. Since we (usually) want to predict numerical SLO values the prediction model needs to be a regression model [3]. We consider the regression model to be a black-box function which takes a list of numeric and nominal values as input, and produces a numeric output (Figure 4). Generally, our approach is agnostic of how this is actually implemented. In our prototype we use multilayer perceptrons (a powerful variant of neural networks) to implement the regression model. Multilayer perceptrons are trained iteratively using a back-propagation technique (maximization of the correlation between the actual outcome of training instances and the outcome that the network would predict on those instances), and can (approximately) represent any relationship between input data and outcome (unlike simpler neural network techniques such as the perceptron, which cannot distinguish data which is not separable by a hyperplane [7]). If a non-numerical SLO should be predicted, a different technique suitable for classification (as opposed to regression) needs to be used to implement the prediction model, e.g., decision trees such as C4.5 [8].

[6] `http://www.cs.waikato.ac.nz/~ml/weka/arff.html`

3.3 Evaluation of Predictors

Another important task of the prediction manager is quality management of predictors, i.e., continually supervising how predictions compare to the actual SLO values once the instance is finished. Generally, we use three different quality metrics to measure the quality of predictions in checkpoints, which are summarized in Table 2. The first metric, *Training Data Correlation*, is a standard machine learning approach to evaluating regression models. We use it mainly to evaluate freshly generated models, when no actual predictions have yet been carried out. This metric is defined as the statistical correlation between all training instance outcomes and the predictions that the model would deliver for these training instances. The definition given in the table is the standard statistical definition of the correlation coefficient between a set of predicted values P and a set of measured values M. However, note that this metric is inherently overconfident in our case, since during training all estimates are replaced for the facts that they estimate (i.e., the training is done as if all estimates were perfect). Therefore, we generally measure the prediction error later on, when actual estimates are being used. However, a low training data correlation is an indication that important facts are still unknown in the checkpoint, i.e., that the checkpoint may be too early.

Name	**Definition**
`Training Data Correlation`	$corr = \frac{cov(P,M)}{\sigma_p \sigma_m}$
`Mean Prediction Error`	$\bar{e} = \frac{\sum_{i=0}^{n} \lvert m_i - p_i \rvert}{n}$
`Prediction Error Standard Deviation`	$\sigma = \sqrt{\frac{\sum_{i=0}^{n} (e_i - \bar{e})^2)}{n}}$

Table 2: Predictor Quality Metrics

This can be done using the *Mean Prediction Error* $\bar{e}$, which is the average (Manhatten) difference between predicted and monitored values. In the definition in Table 2, n is the total number of predictions, p_i is a predicted value, and m_i is the measured value to prediction p_i. Finally, we use the *Prediction Error Standard Deviation* (denoted here simply as σ) to describe the variability of the prediction error (i.e., high σ essentially means that the actual error for an instance can be much lower or higher than $\bar{e}$). In the definition, e_i is the actual prediction error for a process instance ($m_i - p_i$). These metrics are mainly used to give the user an estimation of how trustworthy a given prediction is. Additionally, the `on error` retraining strategy triggers on $\bar{e}$ exceeding a certain threshold.

4 Tool Implementation

In order to verify our approach we built a prototype prediction tool in the Java programming language. Our core implementation is based on our earlier work on event-based monitoring and analysis (as presented in [6] and [4]). Data persistence is provided using a simple MySQL[7] database and Hibernate[8]. We have integrated two different approaches to QoS monitoring: firstly, QoS data as provided by the event-based QoS monitoring approach discussed in [6], and secondly, the QoS data provided by server- and client-side VRESCo [9] QoS monitors [5]. In order to enable event-based monitoring we have used Apache ActiveMQ[9] as JMS middleware. Finally, as has already been discussed, we use the open-source machine learning toolkit WEKA to build prediction models. WEKA is integrated in our system using the WEKA Java API. In addition to the actual prediction tool we have also prototypically implemented the illustrative example as presented in Section 2, as a testbed to verify our ideas (this will be discussed in more detail in Section 5). We have used the WS-BPEL engine Apache ODE, mainly because of ODE's strong support for BPEL lifecycle events. We have also set up the necessary base services which are used in the example (e.g., supplier services, banking service, stock service) using Apache CXF[10].

```
1  <cpdl:checkpoints
2    xmlns:cpdl="http://www.infosys.tuwien.ac.at/2009/cpdl">
3
4    <checkpoint
5      name="beforeGetPaymentPrefs"
6      activityName="getPaymentPrefs" breakBefore="true"
7      predictor="weka.classifiers.functions.MultilayerPerceptron">
8
9      <update type="periodically" value="5"/>
10     <class ppmRef="ORDER_FULFILLMENT_LEAD_TIME"/>
11     <fact ppmRef="RESPONSE_TIME_WAREHOUSE"/>
12     <fact ppmRef="ORDER_INSTOCK"/>
13     <!-- more facts -->
14     <estimate name="getPaymentPrefsResponseTime" type="integer">
15       <estimatorClass
16         class="at.ac.tuwien.infosys.branimon.VrescoQoSEstimator"/>
17       <argument value="ResponseTime"/>
18       <argument value="CustomerService"/>
19       <estimatedField ppmRef="RESPONSE_TIME_GETPAYMENTPREFS"/>
20     </estimate>
21
22     <!-- more estimates -->
23   </checkpoint>
24
25 <checkpoints>
```

Fig. 5: Checkpoint Definition in XML Notion

[7] http://www.mysql.com/
[8] https://www.hibernate.org/
[9] http://activemq.apache.org/
[10] http://cxf.apache.org/

As discussed in Section 3, the main input for our approach is a list of checkpoint definitions. In our current prototype, these definitions are given in a proprietary XML-based language, which we refer to as CPDL (Checkpoint Definition Language). An exemplary excerpt can be seen in Figure 5. In the figure, a checkpoint, which is hooked before the execution of the invoke activity "getPaymentPrefs", is defined. A multilayer perceptron is used as prediction model. The checkpoint will be retrained periodically every 5 hours, and will predict the SLO `ORDER_FULFILLMENT_LEAD_TIME`. Then a number of available facts and estimates are specified. For estimates, an estimator class is given as a full qualified Java class name, which implements the actual prediction. Additionally, a number of arguments can be given to the estimator class. Finally, for every estimate a link to the estimated fact needs to be specified. Note that we do not define facts directly in CPDL. Instead, we reuse the model presented in [6], where we discussed a language for definition of facts using calculation formulae and XLink[11] pointers to WS-BPEL processes (so-called PPMs, process performance metrics). In CPDL, `ppmRef`s are identifiers which point to PPMs in such a model. The complete XML Schema definition of CPDL is available online[12].

5 Experimentation

In order to provide a first validation of the ideas presented we have implemented the illustrative example as discussed in Section 2, and run some experiments using our prototype tool. All experiments have been conducted on a single test machine with 3.0 GHz and 32 GByte RAM, running under Windows Server 2007 SP1. We have repeated every experiment 25 times and averaged the results, to reduce the influence of various random factors such as current CPU load or workload of the process engine.

Instances	Training [ms]
100	3545
250	8916
500	17283
1000	31806

(a) Training Overhead

Instances	Prediction [ms]
100	615
250	630
500	631
1000	647

(b) Prediction Overhead

Table 3: Overhead for Training and Prediction

In Table 3 we have sketched the measured times for two essential operations of our system. Table 3(a) depicts the amount of time in milliseconds necessary to build or refresh a prediction model in a checkpoint. The most important factor here is clearly the time necessary to train the machine learning model,

[11] `http://www.w3.org/TR/xlink/`
[12] `http://www.infosys.tuwien.ac.at/staff/leitner/cpdl/cpdl_model.xsd`

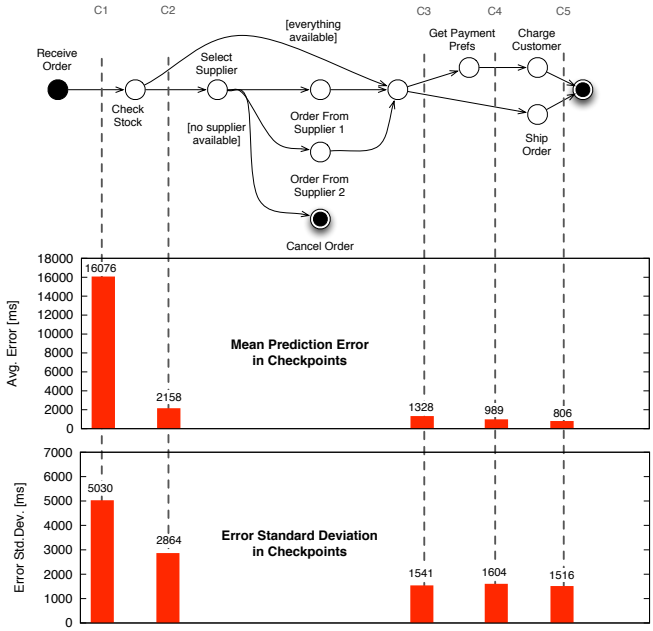


Fig. 6: Prediction Error in Checkpoints

e.g., to train the neural network in our illustrative example. This factor mainly depends on the number of training instances available. In Table 3(a) it can be seen that the time necessary for building the model depends linearly on the number of historical instances available. However, even for e.g., 1000 instances the absolute rebuilding time is below 32 seconds, which seems acceptable for practice, considering that model rebuilding can be done sporadically and offline. Additionally, when rebuilding the model, there is no time where no prediction model is available at all. Instead, the new model is trained offline, and exchanged for the last model as soon as training is finished. A more detailed discussion of these factors is out of scope of this paper for reasons of brevity. In Table 3(b) we have sketched the time necessary for actual prediction, i.e., the online part of the system. As can be seen this overhead is constant and rather small (well below 1 second), which seems very acceptable for prediction at run-time.

Even more important than the necessary time is the accuracy of predictions. To measure prediction accuracy, we have realized five checkpoints in the illustrative example (see top of Figure 6): C1 is located directly after the order is received, C2 after the internal warehouse is checked, C3 after eventual orders from external suppliers have been carried out, C4 during the payment and shipment process, and finally C5 when the execution is already finished. In each of those checkpoints we have trained a prediction model using 1000 historical process instances, and have specified all available data as facts. For not yet available

QoS metrics we have used the average of all previous invocations as estimate. Missing instance data has been treated as unknown. We have used each of those checkpoints to predict the outcome of 100 random executions, and calculated the Mean Prediction Error $\bar{e}$ and the Error Standard Deviation σ (both as defined in Section 3). As expected, $\bar{e}$ is decreasing with the amount of factual data available. In C1, the prediction is mostly useless, since no real data except the user input is available. However, in C2 the prediction is already rather good. This is mostly due to the fact that in C2 the information whether the order can be delivered directly from stock is already available. In C3, C4 and C5 the prediction is continually improving, since more actual QoS facts are available, and less estimates are necessary. Speaking in absolute values, $\bar{e}$ in e.g., C3 is 1328 ms. Since the average SLO value in our illustrative example was about 16000 ms, the error represents only about 8% of the actual SLO value, which seems satisfactory. Similar to $\bar{e}$, σ is also decreasing, however, we can see that the variance is still rather high even in C3, C4 and C5. This is mostly due to our experimentation setup, which included the (realistic) simulation of occasional outliers, which are generally unpredictable.

6 Related Work

The work presented in this paper is complementary to the more established concept of SLA management [10]. SLA management incorporates the definition and monitoring of SLAs, as well as the matching of consumer and provider templates. [10] introduces SLA management based on the WSLA language. However, other possibilities exist, e.g., in [11] the Web Service Offerings Language (WSOL) has been introduced. WSOL considers so-called Web service offerings, which are related to SLAs. Runtime management for WSOL, including monitoring of offerings, has been described in [12], via the WSOI management infrastructure. In our work we add another facet to this, namely the prediction of SLA violations before they have actually occurred. Inherently, this prediction demands for some insight into the internal factors impacting composite service performance. In [13], the MoDe4SLA approach has been introduced to model dependencies of composite services on the used base services, and to analyze the impact that these dependencies have. Similarly, the work we have presented in [4] allows for an analysis of the impact that certain factors have on the performance of service compositions. SLA prediction as discussed in this paper has first been discussed in [14], which is based on some early work of HP Laboratories on SLA monitoring for Web services [15]. In [14], the authors introduced some concepts which are also present in our solution, such as the basic idea of using prediction models based on machine learning techniques, or the trade-off between early prediction and prediction accuracy. However, the authors do not discuss important issues such as the integration of instance and QoS data, or strategies for updating prediction models. Additionally, this work does not take estimates into account, and relatively little technical information about their implementation is publicly available. A second related approach to QoS prediction has been pre-

sented recently in [16]. In this paper the focus is on KPI prediction using analysis of event data. Generally, this work exhibits similar limitations as the work described in [14], however, the authors discuss the influence of seasonal cycles on KPIs. This facet has not been examined in our work, even though seasons can arguably be integrated easily in our approach as additional facts.

7 Conclusions

In this paper we have presented an approach to runtime prediction of SLA violations. Central to our approach are checkpoints, which define concrete points in the execution of a composite service at which prediction has to be carried out, facts, which define the input of the prediction, and estimates, which represent predictions about data which is not yet available in the checkpoint. We use techniques from the area of machine learning to construct regression models from recorded historical data to implement predictions in checkpoints. Retraining strategies govern at which times these regression models should be refreshed. Our Java-based implementation uses the WEKA Machine Learning framework to build regression models. Using an illustrative example we have shown that our approach is able to predict SLO values accurately, and does so in near-realtime (with an delay of well below 1 second).

As part of our future work we plan to extend the work presented here in three directions. Firstly, we want to improve the usability of our prototype by improving the GUI, especially with regard to the definition of checkpoints. Currently, this is mostly done on XML code level, which is clearly unsuitable for the targeted business users. Instead, we plan to incorporate a template-based approach, where facts and estimates are as far as possible generated automatically. Secondly, we want to generalize the ideas presented in this paper so that they are also applicable to aggregated SLOs, such as "Average Response Time Per Day". Thirdly, we plan to extend our prototype to not only report possible SLA violations to a human user, but to actively try to prevent them. This can be done by triggering adaptations in the service compositions, for instance using BPEL'n'Aspects [17]. However, more research needs to be conducted in order to define models of how possible SLA violations can best be linked to adaptation actions, i.e., how to best define which adaptations are best suited to prevent which violations.

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).

References

1. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer **11** (2007)

2. Menascé, D.A.: Qos issues in web services. IEEE Internet Computing **6**(6) (2002) 72–75
3. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2 edn. Morgan Kaufmann (2005)
4. Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Leymann, F., Dustdar, S.: Monitoring and Analyzing Influential Factors of Business Process Performance. In: EDOC'09: Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference. (2009)
5. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection . In: MW4SOC 2009: Proceedings of the 4rd International Workshop on Middleware for Service Oriented Computing. (2009)
6. Wetzstein, B., Strauch, S., Leymann, F.: Measuring Performance Metrics of WS-BPEL Service Compositions. In: ICNS'09: Proceedings of the Fifth International Conference on Networking and Services, IEEE Computer Society (2009)
7. Haykin, S.: Neural Networks and Learning Machines: A Comprehensive Foundation. 3 edn. Prentice Hall (2008)
8. Quinlan, J.R.: Improved Use of Continuous Attributes in C4.5. Journal of Artificial Intelligence Research **4** (1996) 77–90
9. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. Technical report, TUV-1841-2009-03, Vienna University of Technology (2009)
10. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web Services on Demand: WSLA-Driven Automated Management. IBM Systems Journal **43**(1) (2004) 136–158
11. Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., Ma, W.: Management applications of the web service offerings language (wsol). Information Systems **30**(7) (2005) 564–586
12. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web Service Offerings Infrastructure (WSOI) – A Management Infrastructure for XML Web Services. In: NOMS'04: Proceedings of the IEEE/IFIP Network Operations and Management Symposium. (2004) 817–830
13. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.C.: Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In: SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing, Washington, DC, USA, IEEE Computer Society (2008) 21–29
14. Castellanos, M., Casati, F., Dayal, U., Shan, M.C.: Intelligent Management of SLAs for Composite Web Services. In: DNIS 2003: Proceedings of the 3rd International Workshop on Databases in Networked Information Systems. (2003) 28–41
15. Sahai, A., Machiraju, V., Sayal, M., Moorsel, A.P.A.v., Casati, F.: Automated SLA Monitoring for Web Services. In: DSOM '02: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, London, UK, Springer-Verlag (2002) 28–41
16. Zeng, L., Lingenfelder, C., Lei, H., Chang, H.: Event-Driven Quality of Service Prediction. In: ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing, Berlin, Heidelberg, Springer-Verlag (2008) 147–161
17. Karastoyanova, D., Leymann, F.: BPEL'n'Aspects: Adapting Service Orchestration Logic. In: ICWS 2009: Proceedings of 7th International Conference on Web Services, Los Angeles, CA, USA, IEEE (2009)

G Daios: Efficient Dynamic Web Service Invocation

Daios: Efficient Dynamic Web Service Invocation

Systems based on the service-oriented architecture (SOA) paradigm must be able to bind to arbitrary Web services at runtime. However, current service frameworks are predominantly used through precompiled service-access components, which are invariably hard-wired to a specific service provider. The Dynamic and Asynchronous Invocation of Services framework is a message-based service framework that supports SOA implementation, allowing dynamic invocation of SOAP/WSDL-based and RESTful services. It abstracts from the target service's internals, decoupling clients from the services they use.

Philipp Leitner,
Florian Rosenberg,
and Schahram Dustdar
Vienna University of Technology

Software systems built on top of service-oriented architectures (SOAs)[1] use a triangle of three operations — publish, find, and bind — to decouple roles participating in the system. Publish and find put requirements on the service registry and the interface definition language. To publish services, an expressive and extensible service definition language must be available and supported by the service registry.[2] The bind operation, however, is independent from the service registry and is handled by the service consumer. In a SOA, consumers must be able to connect to any service they discover during the find step. In addition, they must be able to change this binding at any time (specifically, at runtime) if the original target service becomes unavailable or if the find operation discovers services delivering a more appropriate quality of service level.

Currently, application developers generate *stubs* (service access components, which are typically compiled from a formal service description such as the Web Services Description Language [WSDL]) to invoke services. These stubs handle the actual invocation but are specific to a service provider. If the application invokes a similar service from a different provider, it must regenerate the stubs because services from different providers in the real world never look quite the same. Even if the services provide similar functionality, they usually differ

1089-7801/09/$25.00 © 2009 IEEE

Related Work in Web Service Invocation Frameworks

The Apache Web Services Invocation Framework (WSIF; http://ws.apache.org/wsif) was the first Java-based Web service framework to incorporate dynamic service invocation. The WSIF dynamic invocation interface is intuitive to use if the client application knows the signature of the WSDL operation to invoke. This is an unacceptable precondition for loosely coupled service-oriented architectures (SOAs). Client applications shouldn't have to know service internals such as the concrete operation name. In addition, WSIF provides notoriously weak support for complex XML Schema types such as service parameters or return values. An application can use complex types only if they're mapped to an existing Java object beforehand, which is frequently impossible in dynamic invocation scenarios. These problems, together with the fact that the framework hasn't been under active development since 2003 and the relatively bad runtime performance, render WSIF outdated.

The Apache Axis 2 (http://ws.apache.org/axis2) framework incorporates more SOA concepts than WSIF. It supports client-side asynchrony and works more on a document level than the strictly RPC-based WSIF. Although Axis 2 is still grounded on the use of client-side stubs, it also supports dynamic invocations through the `OperationClient` or `ServiceClient` APIs. However, these interfaces expect the client application to create the invocation's entire payload (for example, the SOAP body) itself. In that case, Axis 2 does little more than transfer the invocation to the server. We expect a higher level of abstraction from a Web service framework for constructing SOA clients. Still, the Axis 2 SOAP and Representational State Transfer (REST) stacks are well developed and high performing. We therefore created an Axis 2 service back end as part of our Dynamic and Asynchronous Invocation of Services (Daios) prototype. The Axis 2 back end uses Daios's dynamic invocation abstraction, but the Axis 2 service stack performs the actual invocation.

Similar problems arise with other recently introduced service frameworks, such as Codehaus XFire (http://xfire.codehaus.org) or XFire's successor, Apache CXF (http://cxf.apache.org). Ultimately, all of these client-side frameworks rely on static components to access Web services, with little to no support for truly dynamic invocation scenarios.

The Java API for XML-based Web services is the latest Java-based Web service specification. JAX-WS, described in Java specification request (JSR) 224,[1] is the official follow-up to JAX-RPC.[2] JAX-WS is implemented, for instance, in the Apache CXF project, where it exhibits problems similar to Apache CXF. Although the name change suggests that JAX-WS is less RPC-oriented than its predecessor, the specification still focuses on WSDL-to-operation mappings, ignoring the messaging ideas of SOA and Web services. JSR 224 doesn't explicitly discuss REST, despite its claims to generally handle XML-based Web services in Java.

Shinichi Nagano and his colleagues introduce a different approach to dynamic service invocation.[3] They bind static stubs to generic instead of precise interfaces. Doing so lets them use the same stubs to invoke any service with a similar interface, thereby enabling looser coupling between client and provider. This approach (unlike ours) can achieve static type safety. It has considerable disadvantages, however. The concept is only feasible for Web services defined using a formalized XML interface (few REST-based services have such interfaces), and the practical implementation of more generic interfaces is often a hard problem, requiring a lot of domain knowledge. Creating a generic framework that SOA clients can use in any problem domain is therefore difficult using this approach.

References

1. D. Kohlert and A. Gupta, "Java API for XML-Based Web Services, Version 2," 2007; http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html.
2. JSR-101 Expert Group, "Java API for XML-Based RPC, Version 1.1," 2003; http://java.sun.com/xml/downloads/jaxrpc.html#jaxrpcspec10.
3. S. Nagano et al., "Dynamic Invocation Model of Web Services Using Subsumption Relations," *Proc. IEEE Int'l Conf. Web Services* (ICWS 04), IEEE CS Press, 2004, p. 150.

in technical details (such as operations or the data encoding used). It's therefore difficult to implement a SOA-based application using client stubs without falling back to generating and loading stubs at runtime (for example, using reflection facilities). We consider such a solution to be a workaround, which further demonstrates the need for stubless service invocation in SOA scenarios.

Our message-based Dynamic and Asynchronous Invocation of Services (Daios) framework lets application developers create stubless and dynamic service clients that aren't strongly coupled to a specific service provider. Instead, Daios's dynamic interface offers a high degree of provider transparency that lets applications exchange service providers at runtime.

Dynamic Service Invocation

Dynamic binding isn't easy with current Web service client frameworks such as Apache Axis 2 or the Apache Web Services Invocation Framework (WSIF). These frameworks rely on client-side stubs to invoke services, which are usually autogenerated at design time. (See the "Related Work in Web Service Invocation Frameworks"

sidebar for a more detailed discussion of these and other current frameworks.) However, stubs are invariably hardwired to a specific service provider and can't be changed at runtime. If service providers are hardwired into the service consumers' application code, producers and consumers can't be considered loosely coupled. The use of client stubs doesn't follow the SOA ideas because the developer performs both find and bind. A client application relying on precompiled stubs can't implement a SOA. We therefore conclude that the SOA triangle is broken.[2]

In addition, Web service client frameworks such as Apache Axis 2 and Apache WSIF often suffer from a few further misconceptions. They're often built to be as similar as possible to earlier distributed object middleware systems,[3] implying a strong emphasis on RPC-centric and synchronous Web services. SOAs, on the other hand, center on the notion of synchronous and asynchronous exchange of business documents.

We define several requirements for a Web service invocation framework that supports the core SOA ideas.

The first requirement is *stubless service invocation*. Given that generated stubs entail a tight coupling of service provider and service consumer, the invocation framework shouldn't rely on static components such as client-side stubs or data transfer objects. Instead, the framework should be able to invoke arbitrary Web services through a single interface using generic data structures.

Second, a Web service invocation framework should be *protocol independent*. Web service standards and protocols are not yet fully settled. Discussion continues about the advantages of the Representational State Transfer (REST)[4] architecture compared to the more common SOAP and WSDL-based[5,6] approaches to Web services. The framework should therefore be able to abstract from the underlying Web service protocol and support at least SOAP- and REST-based services as transparently as possible.

Third, the framework must be *message driven*. Web services are often seen as collections of platform-independent remote methods. The framework must be able to abstract from this RPC style, which usually leads to tighter coupling, and follow a message-driven approach instead. Additionally, the message-driven interface should be as simple as possible to facilitate the creation of complex messages.

Next, the framework should support *asynchronous communication*. In a SOA, services might take a long time to process a single request. The prevalent request-response communication style is unsuitable for such long-running transactions. The framework should therefore support asynchronous (nonblocking) communication.

Fifth, it must provide *acceptable runtime behavior*. The framework shouldn't imply sizable overhead on the Web service invocation. Using the framework shouldn't take significantly longer than using any of the existing Web service frameworks.

Unfortunately, current Web service frameworks don't fully live up to these requirements.

The Daios Solution

Given our requirements for a Web services invocation framework, we designed the Daios framework and implemented a system prototype. Daios is a Web service invocation front end for SOAP/WSDL-based and RESTful services. It supports fully dynamic invocations without any static components such as stubs, service endpoint interfaces, or data transfer objects.

Figure 1 sketches the Daios framework's general architecture. It also shows where the general SOA triangle of publish, find, and bind fits into the framework. The framework consists of three functional components:

- the general Daios classes, which orchestrate the other components;
- the interface parsing component, which preprocesses the service description (for example, WSDL and XML Schema); and
- the service invoker, which uses a SOAP or REST stack to conduct the actual Web service invocations.

Clients communicate with the framework front end using Daios messages (a Daios-specific message representation format). The framework's general structure is an implementation of the composite pattern for stubless Web service invocation (CPWSI).[7] CPWSI separates the framework's interface from the actual invocation back-end implementation and allows for flexibility and adaptability.

Daios is grounded on the notion of message exchange. Clients communicate with services by passing messages to them. Services return the invocation result by answering with mes-

sages. Daios messages are potent enough to encapsulate XML Schema complex types but are still simpler to use than straight XML. Messages are unordered lists of name-value pairs, referred to as *message fields*. Every field has a unique name, a type, and a value. Valid types are either built-in types (simple field), arrays of built-in types (array field), complex types (complex field), or arrays of complex types (complex array field). Such complex types can be constructed by nesting messages. Users can therefore easily build arbitrary data structures without needing a static type system.

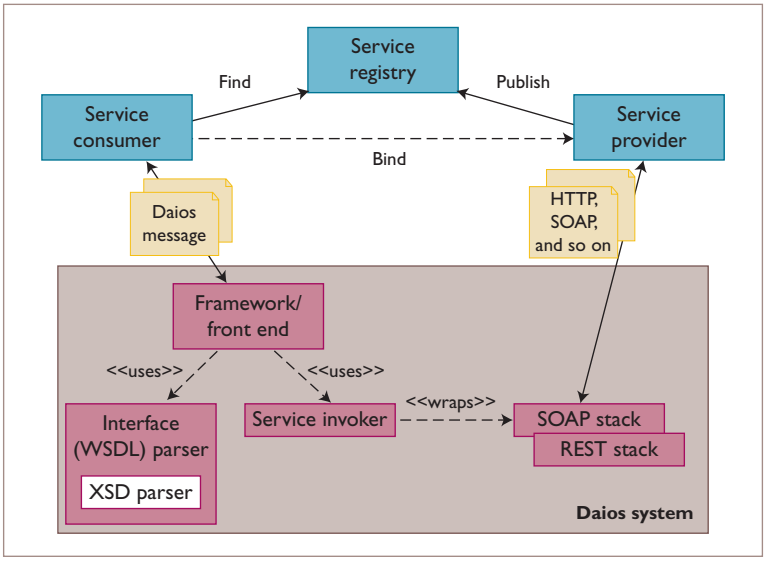


Figure 1. The Dynamic and Asynchronous Invocation of Services (Daios) framework's overall architecture. The framework supports the service-oriented architecture publish, find, and bind paradigm.

Invoking Services with Daios

Using Daios is generally a three-step procedure:

First, clients find a service they want to invoke (service discovery phase). The service discovery problem is mostly a registry issue and is handled outside of Daios.[2]

Next, the service must be bound (preprocessing phase). During this phase, the framework collects all necessary internal service information. For example, for a SOAP/WSDL-based service, the service's WSDL interface is compiled to obtain endpoint, operation, and type information.

The final step is the actual service invocation (dynamic invocation phase). During this phase, Daios converts the user input message into the encoding expected by the service (for instance, a SOAP operation for a WSDL/SOAP-based service, or an HTTP `get` request for REST), and launches the invocation using a SOAP or REST service stack. When the service stack receives the invocation response (if any), it converts it back into an output message and returns it to the client.

Once a service is successfully bound, clients can issue any number of invocations without having to rebind. Service bindings must be renewed only if the service's interface contract changes or the client explicitly releases the binding.

Most of Daios's important processing occurs in the dynamic invocation phase. For a SOAP invocation, the framework analyzes the given input and determines which WSDL input message the provided data best matches. For this, Daios relies on a similarity algorithm. This algorithm calculates a structural distance metric for the WSDL message and the user input — that is, how many parts in a given WSDL message have no corresponding field in the Daios message, where lower values represent a better match. For fields in the user message with no corresponding field in the WSDL message, the similarity is ∞. Daios invokes the operation whose input message has the best (that is, lowest) structural distance metric to the provided data. If two or more input messages are equally similar to the input, the user must specify which operation to use. If no input message is suitable — that is, if all input messages have a similarity metric of ∞ to the input — an error is thrown. Here, the provided input is simply not suitable for the chosen Web service. Otherwise, the framework converts the input into an invocation of the chosen operation, issues the invocation, receives the result from the service, and converts the result back into a message.

The back end used to conduct the actual invocation is replaceable. The Daios research prototype offers two invocation back ends. One uses the Apache Axis 2 stack, the other uses a custom-built (native) SOAP and REST stack. Daios emphasizes client-side asynchrony. All invocations can be issued in a blocking or nonblocking fashion.

This procedure abstracts most of the RPC-like internals of SOAP and WSDL. The client-side application doesn't need to know about WSDL operations, messages, end points, or encoding. Even whether the target service is implemented as a SOAP- or REST-based service is somewhat transparent to the client, although for REST services, clients need to know

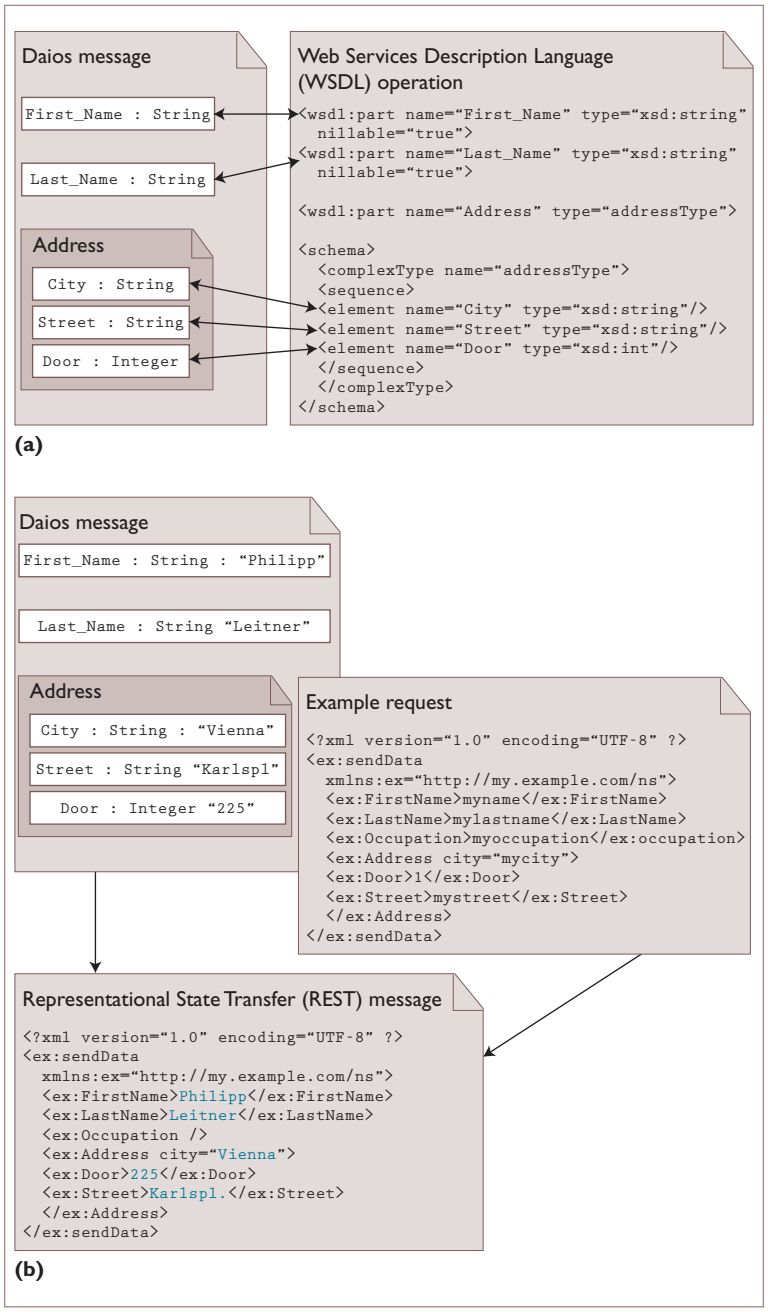


Figure 2. Matching Daios inputs to service interfaces. (a) A Daios message and a WSDL operation with a structural distance of 0. (b) When Daios receives an invocation, it fills the template (example request) with user input to produce a valid REST invocation.

the endpoint address. Daios handles all of these service details, so the client application can be as generic as possible. The service client needs to know only the names and types of mandatory service parameters (for example, WSDL operation parameters).

For REST invocations, the user can specify an example request instead of the WSDL interface, or not give further details on the service interface at all. If the user gives an example request, Daios uses the example as a template, which it fills with the user's actual input at invocation time, and issues an HTTP `post` request with the filled template as payload. If no information about the service interface is given (that is, neither WSDL description nor example request), Daios issues an HTTP `get` request with URL-encoded parameters.

Figure 2 shows the matching of Daios inputs to a WSDL description and an example request. Figure 2a shows a Daios message and a WSDL message in RPC/encoded style with a structural distance of 0 (a perfect match). Removing the `First_Name` field from the Daios message increases the structural distance to 1. Figure 2b details how the framework fills an example template with user input provided as a Daios message in a REST invocation.

Usage Examples

The message-based Daios client API is easy to use. Figure 3 shows the Java code necessary to invoke a SOAP/WSDL-based Web service. The message in this example corresponds to the structure depicted in Figure 2. Although the target service uses nested data structures (the registrations contain address data), Daios doesn't need any static components such as data transfer objects. All necessary service and type information is collected during the preprocessing phase (lines 8 to 11). When the actual dynamic invocation is fired (lines 26 and 27), the framework uses this information and converts the user-provided input to a concrete Web service invocation. The example in the figure uses a blocking invocation style, but Daios handles asynchronous communication identically.

The client application doesn't have to specify operation name, endpoint address, or WSDL encoding style used. Daios abstracts this information from the service internals and exposes a uniform interface, allowing loose coupling between client and service.

Figure 4 (p. 78) exemplifies the invocation of a RESTful Web service. In this figure, a client application is accessing the Flickr REST API and retrieving a list of hyperlinks to the most interesting photos.

Daios invokes RESTful and SOAP-based services through the same interface (the code necessary to access the service is practically identical for both Web service types). The main

```
1    // create a Daios backend
2    ServiceFrontendFactory factory = ServiceFrontendFactory.getFactory
3        ("at.ac.tuwien.infosys.dsg.daiosPlugins."+
4        nativeInvoker.NativeServiceInvokerFactory");
5
6    // preprocessing - bind service
7    ServiceFrontend frontend = factory.createFrontend(new URL(
8         "http://vitalab.tuwien.ac.at/"+"orderservice?wsdl"));
9
10   // construct input that we want
11   // to pass to the service
12   DaiosInputMessage registration = new DaiosInputMessage();
13   DaiosMessage address = new DaiosMessage();
14   address.setString("City", "Vienna");
15   address.setString("Street", "Argentinierstrasse");
16   address.setInt("Door", 8);
17   registration.setComplex("Address", address);
18   registration.setString("First_Name", "Philipp");
19   registration.setString("Last_Name", "Leitner");
20
21   // dynamic invocation
22   DaiosOutputMessage response = frontend.requestResponse(registration);
23
24   // retrieve result
25   String regNr = response.getString("registrationNr");
26   // ...
```

Figure 3. A Daios SOAP invocation. The application constructs both a new service front end to the SOAP-based Web service described by a Web Services Description Language contract and an input message in Daios message format, and issues the invocation using a blocking request response invocation.

difference is that no interface definition language similar to WSDL has yet been established for RESTful services, so the user must specify more service details for REST-based invocations (the endpoint address in the example).

Evaluation

We evaluated our prototype against various Web service frameworks: Apache WSIF, Apache Axis 2, Codehaus XFire, and Apache CXF (see the sidebar). We compared the frameworks in terms of supported functionality, response times, and memory consumption. For brevity, we present only functional aspects and runtime performance data in this article.

Table 1 (p. 79) shows how well the candidate frameworks meet our requirements for a Web service invocation framework. We present the last of these requirements — acceptable runtime behavior — later. Current service frameworks fail to meet these requirements in some important respects. The core problem is that none of them embraces SOA's loosely coupled document-centric approach. Rather, they're based on an RPC processing model, demanding explicit knowledge of service internals such as WSDL encoding styles, operation signatures, and endpoint addresses. Additionally, neither WSIF nor XFire provide a fully expressive dynamic invocation interface. User-defined (complex) types are difficult to use over these interfaces if the application doesn't know the types at compile time. Most interfaces support the REST style of Web services, but they don't support a transparent integration of SOAP and REST. The Daios prototype solves all these problems. It exposes a simple messaging interface with which applications can dynamically invoke arbitrary services without knowing the service's implementation details (including whether the service is implemented as a SOAP- or REST-based service), both synchronously and asynchronously.

Figure 5 (p. 79) addresses the acceptable runtime behavior requirement. The figure compares

```
1    String myAPIKey = ... // get an API key from Flickr
2
3    // use the native backend
4    ServiceFrontendFactory factory = ServiceFrontendFactory.getFactory
5        ("at.ac.tuwien.infosys.dsg.daiosPlugins."+
6        nativeInvoker.NativeServiceInvokerFactory");
7
8    // preprocessing for REST
9    ServiceFrontend frontend = factory.createFrontend();
10
11   // setting the EPR is mandatory for REST services
12   frontend.setEndpointAddress(
13     new URL("http://api.flickr.com/services/rest/"));
14
15   // construct message
16   DaiosInputMessage in = new DaiosInputMessage();
17   in.setString("method", "flickr.interestingness.getList");
18   in.setString("api_key", myAPIKey);
19   in.setInt("per_page", 5);
20
21   // do blocking invocation
22   DaiosOutputMessage out = frontend.requestResponse(in);
23
24   // convert WS result back
25   // into some convenient Java format
26   DaiosMessage photos = out.getComplex("photo");
27   // ...
```

Figure 4. A Daios Representational State Transfer (REST) invocation. The application creates a Daios service front end to the Flickr photo service's REST API and retrieves a list of the "most interesting" photos.

the response times of the candidate frameworks in simple SOAP-based Web service invocations. Figure 5a shows the results for RPC/encoded invocations, and Figure 5b shows results for document/literal invocations with wrapped parameters. We only evaluated RPC/encoded invocations for Daios and WSIF. Axis 2, XFire, and CXF don't support this particular WSDL encoding style. Apache WSIF is well behind in both test cases; all other candidate frameworks exhibit similar response times.

We also performed extensive tests using different types of invocations (with binary or array payload data), but the general result was similar for all tests. Additionally, we've gathered similar results for REST-based invocations. We therefore conclude that using Daios doesn't imply a relevant performance penalty over Apache Axis 2, Apache CXF, or Codehaus XFire, and that our prototype is significantly faster than Apache WSIF.

Increasingly, Web service implementations use policies to describe the service's nonfunctional attributes, such as security policies, transactional behavior, and reliable messaging. Often, these implementations use the Web Services Policy framework.[8] We plan to add WS-Policy support to our framework to support policy-enforced interactions. Furthermore, we'll extend our evaluation of the Daios framework to a more extensive real-life scenario to get a more accurate picture of the implementation's runtime performance and usability in real business applications.

We recently released the first version of our Daios prototype as an open source project using Google Code and are currently working on a .NET port.

Acknowledgments

The European Community's Seventh Framework Program (FP7/2007-2013) helped fund the research leading to the results reported here under grant agreement 215483 (S-Cube).

Table 1. Functional comparison of current Web service invocation frameworks (WSIFs).

Requirement	Daios	Apache WSIF	Apache Axis 2	Codehaus XFire	Apache CXF
Stubless service invocation					
Simple types	Yes	Yes	Yes	Yes	Yes
Arrays of simple types	Yes	Yes	Yes	Yes	Yes
Complex types	Yes	No	Yes	No	Yes
Arrays of complex types	Yes	No	Yes	No	Yes
Protocol independence					
Transparent protocol integration	Yes	No	No	No	No
SOAP over HTTP support	Yes	Yes	Yes	Yes	Yes
Representational State Transfer support	Yes	No	Yes	No	Yes
Message-driven approach					
Document-centric interface	Yes	No	No	No	No
Transparent handling of service internals	Yes	No	No	No	No
Support for asynchronous communication					
Synchronous invocations	Yes	Yes	Yes	Yes	Yes
Asynchronous invocations	Yes	No	Yes	No	Yes
Simple API					
Simple to use dynamic interface	Yes	Yes	No	Yes	Yes

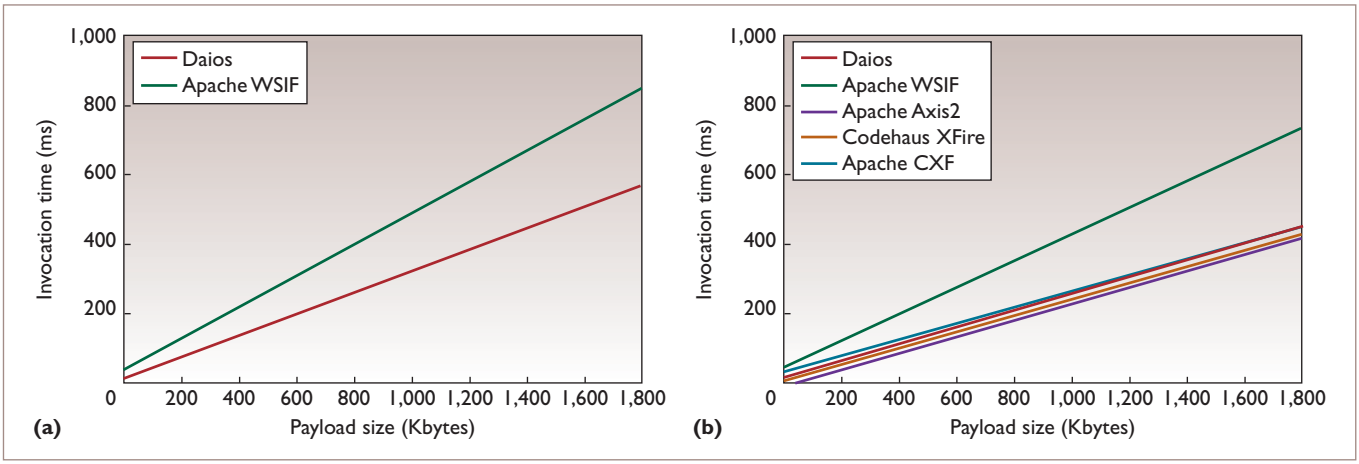


Figure 5. Comparison of invocation response times. (a) RPC/encoded invocations and (b) document/literal invocations with wrapped parameters. Only Daios and the Web Services Invocation Framework (WSIF) support RPC/encoded.

References

1. M.P. Papazoglou et al., "Service-Oriented Computing: State of the Art and Research Challenges," *Computer*, vol. 40, no. 11, 2007, pp. 38–45.
2. A. Michlmayr et al., "Towards Recovering the Broken SOA Triangle: A Software Engineering Perspective," *Proc. 2nd Int'l Workshop on Service Oriented Software Eng.* (IW-SOSE 07), ACM Press, 2007, pp. 22–28.
3. W. Vogels, "Web Services Are Not Distributed Objects," *IEEE Internet Computing*, vol. 7, no. 6, 2003, pp. 59–66.
4. R.T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, doctoral dissertation, Information and Computer Science Dept., Univ. of California, Irvine, 2000.
5. *SOAP Version 1.2 Part0: Primer*, World Wide Web Consortium (W3C) recommendation, 2003; www.w3.org/TR/soap12-part0.
6. *Web Services Description Language (WSDL) Version 2.0 Part0: Primer*, World Wide Web Consortium (W3C) candidate recommendation, 27 Mar. 2006; www.w3.org/TR/2006/CR-wsdl20-primer-20060327.

7. P. Buhler et al., "Preparing for Service-Oriented Computing: A Composite Design Pattern for Stubless Web Service Invocation," *Proc. Int'l Conf. Web Eng.*, LNCS 3140, Springer, 2004, p. 763.
8. J. Schlimmer et al., "Web Services Policy Framework (WS-Policy)," joint specification by IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, and VeriSign, 2006; www.ibm.com/developerworks/library/specification/ws-polfram.

Philipp Leitner is a PhD student in the Distributed System Group at the Vienna University of Technology. His research interests include general issues of distributed computing, especially service-oriented computing, service-oriented architectures, Web services, peer-to-peer computing, and network management. Leitner has a master's degree in business informatics from the Vienna University of Technology. Contact him at leitner@infosys.tuwien.ac.at.

Florian Rosenberg is a PhD candidate in the Distributed System Group at the Technical University Vienna. His research interests include software composition, service-oriented architectures, and software engineering. Rosenberg has a master's degree in software engineering from the Upper Austria University of Applied Sciences. Contact him at florian@infosys.tuwien.ac.at.

Schahram Dustdar is a full professor of computer science, director of the Vienna Internet Technologies Advanced Research Lab, head of the Distributed Systems Group of the Information Systems Institute at the Vienna University of Technology, and honorary professor of information systems in the Department of Computer Science at the University of Groning, the Netherlands. His research interests include service-oriented architectures and computing, mobile and ubiquitous computing, complex and adaptive systems, and context-aware computing. Dustdar has a PhD in business informatics from the University of Linz, Austria. He's a member of the IEEE Computer Society and the ACM. Contact him at dustdar@infosys.tuwien.ac.at.

H Dynamic Adaptation of the Master-Worker Paradigm

Dynamic Adaptation of the Master-Worker Paradigm

Françoise André, Guillaume Gauvrit, Christian Pérez
IRISA / INRIA
Campus de Beaulieu, 35042 Rennes cedex, France
{Francoise.Andre, Guillaume.Gauvrit, Christian.Perez}@irisa.fr

***Abstract*—The size, heterogeneity and dynamism of the execution platforms of scientific applications, like computational grids, make using those platforms complex. Furthermore, today there is no effective and relatively simple solution to the programming of these applications independently of the target architectures. Using the *master-worker* paradigm in software components can provide a high level abstraction of those platforms, in order to ease their programming and make these applications portable. However, this does not take into account the dynamism of these platforms, such as changes in the number of processors available or the network load.**

Therefore we propose to make the master-worker abstraction dynamically adaptable. More specifically, this paper characterizes the master-worker paradigm on distributed platforms, then describes a decision algorithm to switch between master-worker implementations at run-time.

Keywords-dynamic adaptation; master-worker paradigm; software engineering; grid computing;

I. Introduction

Distributed systems range from single computers with a multicore processor to grids that aggregates many computer clusters. They are usually composed of many computers of various kind that are connected by heterogeneous networks. Their size and heterogeneity makes them complex to program. Besides, the variability of resources and their sharing among users make these systems dynamic, especially computational grids. Indeed, new computers may be added to the grid while some may go to maintenance, and the workload as well as the network load can vary significantly during the execution of a task. Additionally, the applications developed for these systems are increasingly complex and thus hard to design. Today, there is no efficient and relatively simple solutions to program applications for distributed systems regardless of the target infrastructure.

These distributed systems are greatly used to run scientific or engineering applications that need a large amount of computing power. For instance in molecular biology to fold proteins, or in avionics to study air flow. Many of these applications are parametric ones, where different instances of the same code are executed in parallel on varying parameters. These parametric applications can be well designed by using the master-worker paradigm, hence the number of master-worker software or middleware one can encounter like SETI@Home [1], NetSolve [2] or DIET [3]. So, in this paper, we focus only on master-worker software aimed to be run on distributed systems. Some of these middleware are well suited for low scale applications while others are a better fit for high scale or highly dynamic infrastructures. Today, one has to choose at design time between all these alternatives to develop a master-worker application, and one cannot switch easily between them when developing, even less at run time.

So, to deal with these difficulties, in the aim to ease the development of parametric applications for distributed systems, we study in this paper how to dynamically adapt the master-worker paradigm, focusing on the creation of an algorithm to decide when to adapt. Section II presents various master-worker implementations. Section III describes a way to adapt the master-worker paradigm. Then Section IV characterizes master-worker applications on distributed systems to build foundations for an algorithm to make adaptation choices, which is described in Section V Finally, a conclusion ends this paper.

II. Master-Worker implementations

The software or middleware implementing the master-worker paradigm is usually composed of five parts: the *master* sends tasks to compute to *workers*, *monitors* collect information about the state of the hardware and software resources which is stored in a *database*, and a *scheduler* selects to which workers the tasks are to be sent according to their kind and the informations stored in the database.

Many alternatives can be used to implement the master-worker paradigm, each better fitted to different situations.

The obvious one is to hard-code the paradigm in the application, with the scheduler and the database into the master. The worker would be chosen according to the *round-robin* pattern, that is one after the other, cycling. No monitor would be used. One drawback of this approach is the poor separation of concern; one other is that the dynamism of the grid or cluster is not at all taken into account. However, this implementation can be fast, simple to implement and well suited to software to be run on private clusters (not shared) or for prototypes.

Another alternative is to use a framework, like NetSolve [2]. It is composed by a *master*, an *agent* and *workers*, as shown on Figure 1, where each worker should be deployed on a different computer. Here, the *scheduler* and the *database* are in the agent, while the workers can *monitor* the speed and the load of their computer as well as the latency and bandwidth of the network between them and the agent. This information is sent

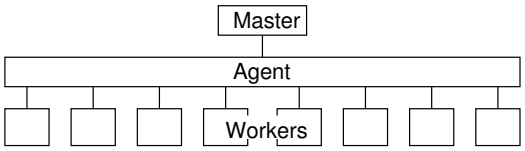


Fig. 1. NetSolve

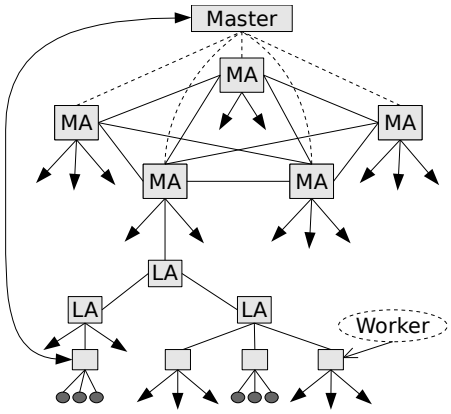


Fig. 2. DIET

to the agent just after the deployment of the workers. When the master has to send a request, it asks the agent for a list of workers able to compute the task, then the master tests the delay to the workers and sends the request to the fastest one.

This kind of centralized implementation of the paradigm solves the problem of the separation of concerns and it can in part deal with the dynamism of the resources. Nevertheless, this centralized architecture may not be well suited to large distributed systems, like large grids.

DIET [3] is specifically developed for those large distributed systems. As shown on Figure 2, DIET is composed by four elements: *masters*, *master agents* (MA), *local agents* (LA), and *workers*. The schedulers are only in the MAs, and the monitors at the level of the workers. The agents should be distributed according to the network topology, for example one MA by grid and one LA by cluster. The LAs are used to relay requests and information between MAs and workers. A LA stores various local information useful to distribute requests. However, the costs of this implementation can be significant for software using very short tasks.

NINF [4] and Nimrod/G [5] can be cited among other available frameworks.

The differences among these implementations makes them fit differently to various situations. For example, when an application with a dynamic behavior switches from short tasks of about 10 ms, to longer tasks averaging 100 s, it might be beneficial to switch from a simple and fast implementation like round-robin to another one better able to distribute the workload, like NetSolve or DIET. However, this is not currently feasible, unless if this logic is implemented in the application, which breaks the separation of concerns. A better solution would be to use an abstraction of these implementations of the paradigm and let it switch between different implementations when needed.

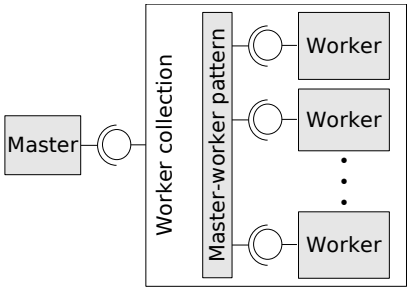


Fig. 3. The master-worker paradigm, in software components

III. Support for the dynamic adaptation

In order to switch between master-worker implementations at run-time, we use an abstraction of the paradigm, presented in [6], well suited to its adaptation: a component-based collection of workers. The principle of a collection abstracts the paradigm from the distributed systems architecture, while the use of *software components* provides the separation of concerns needed to adapt the collection of workers. A *software component* [7] is a software entity, with a well defined behavior , that can be composed with other components using *ports* (either *sender* or *receiver*) for communication between them.

In [6], the authors suggest to represent each master and worker in a component. The developers of the final application could write it as if there would be only one master and one worker connected together; where, in fact, worker components are put together in a collection component, as shown on Figure 3. The collection uses a master-worker pattern to handle the communication between masters and workers and the distribution of the requests. A pattern can use any implementation of the paradigm, like NetSolve or DIET.

Despite that the number of workers and the pattern can be chosen when deploying the application, it was not studied how to do it dynamically, when a need or opportunity arise to change the configuration.

In the article [8], the authors study how to make this collection of workers dynamically adaptable and suggest to use the *Dynaco* [9] framework to adapt it. This framework is to be integrated in the component to adapt; here, the collection component. It divides the dynamic adaptation in four major parts: the *monitoring* of constraints, the *decision*, the *planning* and the *execution*. Thus, to use this framework, one has to connect it to probes which gather the information needed to make a decision and to plan the adaptation. This information is gathered passively by an *observer* interface and actively by a *monitor* interface. Then, one has to provide three components to do the last three functions, as shown in the Figure 4. The *decider* component decides when to adapt and sends a *strategy* to the *planner* component when an adaptation is needed. The *planner* component then breaks down the

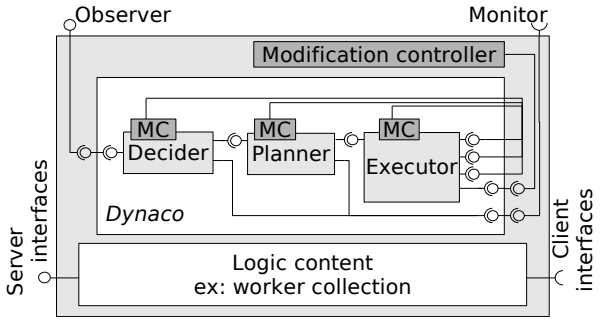


Fig. 4. Dynaco

strategy into an action *plan*, which is a set of elementary tasks. This plan is then sent to the *executor* component which can adapt the collection according to the plan, using *modification controllers*.

In our work, we focused on the *decision* part of the adaptation, which will be presented in Section V But first, Section IV studies what to monitor in order make adaptation choices.

IV. Characterization of the master-worker paradigm on distributed systems

In this study, we assume that the implementation of the paradigm is limited to one master, one worker collection already deployed and one master-worker pattern. The decision algorithm which has to choose when to switch between master-worker patterns needs to monitor the master-worker application and the distributed system in order to have enough information to make its decisions. To discriminate the pertinent parameters among all of the possible ones, we first have to state *why* to adapt and *what* is to adapt.

The aims of the dynamic adaptation of the master-worker paradigm are manifold. In our study, we focused to those defined from an user point of view, in contrast to a computing resources manager. Besides, since it is impossible to be exhaustive, we studied only the following objectives:

- Maximize the execution speed of the application. It is measured by the average number of executed requests by second. We assume that every pattern ensure that every request sent by the master is processed (i.e. there is no starvation).
- Minimize the processing time of any request, independently of the other requests. This objective should not be confused with the preceding one. This objective enables to consider the case where only some of the requests' processing time are to be minimized.
- Respect a maximum time limit to process a request. This objective enables to design "real-time" applications. It is to the user to specify reasonable time limits, i.e. that can be respected, otherwise the respect of this constraint cannot be ensured.

Other objectives such as respect of a maximum cost or minimizing the processing time of a pack of requests, can be thought of, but they are not studied in this paper.

Now that we know why the application have to be adapted, we have to define what is to adapt. Two parameters of the worker collection can be adapted dynamically.

Firstly, the number of workers can be adjusted as the number of simultaneous requests to be processed evolves with the time, while keeping this number inside the boundaries delimited by the hardware. Also, it can be adjusted when computers appear or leave the distributed system.

Secondly, the master-worker pattern can be changed when evolutions of the distributed system or the type of requests being processed make a unused pattern more suitable to these new conditions.

We focused on three representative patterns in our study: *round-robin*, *load-balancing* and a modified version of *DIET*. Experiments were done with these patterns on the platform *Grid'5000* [1]. Results are given in [10].

A worker is said to be *free* when it can handle a new request sent by the master without affecting the possible other requests being processed by the worker. A worker might only be able to handle one request at the same time.

A function that estimate the extra cost in time due to the pattern must be provided for each pattern, for the decision algorithm. This function is called the *extra cost function*. It can be implemented statically, which can be best suited to simple patterns like round-robin or load-balancing, or by using a learning mechanism, which would be better suited to more complex patterns, like DIET.

a) Round-robin: This pattern distributes the number of request equally among the workers.

To this end, the pattern keeps a list of workers in a ring and each worker keeps a queue of requests. Each request sent by the master is transmitted to the worker next to the last who were sent one. Each request received by a worker is put in the queue. The worker process the request in the incoming order and can process one or many requests at a time (for example, one by CPU core).

From the results of the experiments done by Hinde L. Bouziane in her PhD thesis [10], the extra cost function can be approximated by:
$extraCost = a \cdot numberOfWorkers + b$ (in seconds), where $a \approx 9.97 \cdot 10^{-6}$ s and $b \approx 1.62 \cdot 10^{-2}$ s.

b) Load Balancing: This pattern distributes the workload equitably among the workers.

It handles a single queue of requests. When a request is received by the master, it is added to the queue. When a request comes to an empty queue, it is stored if there is no free worker, else it is send randomly among the free workers. When a worker send the result of a request, a request from the master's queue is sent back to it if the queue is not empty.

The extra cost function is the same than round-robin's one with $a \approx 1.46 \cdot 10^{-5}$ s and $b \approx 1.62 \cdot 10^{-2}$ s.

[1] Grid'5000: https://www.grid5000.fr (2008)

c) DIET: This pattern uses a request sequencing policy, it uses a distributed architecture with many agents and has probes to its disposal to estimate the workers' processing speed.

We use a modified version of DIET, which differs from the original by a scheduler which sorts the requests by the estimated time to process the requests, when possible.

A single queue of requests is managed by DIET. If an estimation of the time to process each request is available, the queue is sorted in decreasing order of the estimated lengths (we assume there is no starvation or that a priority mechanism is used to avoid it). When a request is received by the master, it is added to the queue. When a request comes to an empty queue it is stored if there is no free worker; else a request is sent to each free worker to estimate and compare their processing speed, then the pending request is sent to the fastest free worker. When a worker sends the result of a request, if the queue is not empty, the master probes the workers to send a request to the fastest one.

Our extra cost function would require further experimental results to refine it and so is not detailed here.

For round-robin, no mechanism is provided to manage requests having to be processed in a limited time. Whereas for the Load balancing and DIET patterns, an estimation of the time to process (in number of cycles) have to be provided by each request if the user chooses to put to each request a time limit to process it. Furthermore, in this case, every request must have a time limit to avoid a starvation, and the queue is then sorted by limit date to send the requests for them to be executed in time.

In order to be able to decide to adapt, the application needs to monitor itself and the distributed system. To this end, we have to select relevant parameters for the adaptation. We consider a potential parameter as useful if its modification can generate a need to adapt, or if it can be used to describe the state of the application's part to adapt (here, the number of workers and the master-worker pattern).

These parameters have to be measurable or known by the application. The list follows:

Known parameters: They do not need to be measured. They are composed of: the number of workers, the pattern used and the number of requests being processed.

Direct parameters: They have to be measured. They are those like "the time to process a request" or "the workload of each workers" which are needed to compute the *indirect parameters*; and "the number of workers" which can differ from the known parameter, for example in case of a failure.

Indirect parameters: They are to be computed (for conciseness we do not describe how do do it):

- The variability of the workload of the workers due to the environment exterior to the workers,
- The variability of the processing power of the workers due to the execution of requests,
- The variability of the time to execute request,
- The time to transmit a request (sending the request and receiving the results) in function of the data's size to transmit, average bandwidth and average delay (from the master to the workers),
- The heterogeneity of the network
- The maximum number of workers, depending on the number of computers.

V. Decision Algorithm to Change the Pattern

There are at least two ways to adapt the collection. The first is to change the number of workers depending to the workload and the available computing resources. This is not described in this paper for the sake of conciseness. The second way is to dynamically change the master-worker pattern. This paper only deals with this later algorithm.

The decision algorithm is based on the description of the behavior of the patterns, which depends on the state of the pattern and the distributed system, and on a QoS objective.

It is designed to be generic: new patterns can be added to the application without modifying the existing parts of the algorithm implementation. In addition, the algorithm can be adapted once the application is deployed, thanks to Dynaco framework's dynamic adaptability. To this end, the behavior description of a pattern is independent from those of the other patterns.

It is also designed to be added to a learning mechanism which could tune its decisions according to the results of preceding adaptations. However, such mechanism is not compulsory.

The algorithm is a compromise between performance and an ideal solution. Indeed, It does not aim to select the pattern the best fitted to every situation. Instead, it aims to discriminate a pattern among the best fitted in a short time. Moreover, it is not always useful to select the optimal pattern among two almost equivalent ones, as long as the inadequate ones are filtered.

The principle of the algorithm is to compare the patterns using a description of their behavior. This comparison is done using positives scores, including $+\infty$: the pattern with the lowest score is the best fitted to the situation for the given QoS.

The behavior of a pattern don't have to be set for each QoS objectives, but a pattern can only be used when its behavior is set for the selected QoS objective.

The behaviors of the patterns are divided into elementary behaviors for which a *cost function* is defined. Each of these functions is based on a *characteristic* which is built from parameters taken among those presented previously.

The aim is to provide functions representing approximatively the extra cost due to each characteristic. In this sense, there is no need to find the exact functions, knowing that the weighting refines them. Furthermore, since the user might want to use its application to compute very short tasks, it is important for the computation of the indices to be fast, which limits the possible complexity of the functions.

These functions can be discovered using simulations, by monitoring the behavior of the pattern in controlled environment or by knowing how the pattern behaves.

We have identified nine characteristics such as the number of workers, the number of requests being processed, the heterogeneousness of the network.

For each of these characteristics an index is computed by the cost function, that represents the extra cost resulting of the characteristic.

Once the indices are computed, they are weighted and added to make a score by pattern. Then the scores are compared; the pattern with the lowest score is selected. If the difference between this score and the score of the pattern currently used is greater than 5% of the latter (that is, if the difference between the scores is significant), then an adaptation is triggered. If all the scores are infinites, there is no lowest score, so there is no pattern selected and no adaptation triggered. In case of conflict between patterns, one is to be arbitrary selected, the last used might be a good choice to avoid the overhead of deploying a new pattern.

We studied the behavior of the three patterns for the three QoS objectives.

A. Simulations

To study the dynamic adaptation of the master-worker paradigm, we made simulations and used experimental results from work done in our project-team, published in Hinde L. Bouziane's PhD thesis [10].

To validate the decision algorithm, two engines were developed: one to simulate master-worker patterns on distributed systems, the other to take decisions according to the algorithm. The first one is used to simulate the behavior of the various schedulers of the patterns and characteristics of the distributed systems (like the computers workload). The second one offers to simulate the evolution in the time of the parameters used in the decision algorithm, in order to visualize adaptation choices done by the algorithm.

To give a rough idea of the size of these simulators: they are written in Java and range from 1200 to 1500 lines of code. For its decision engine, the simulator of adaptation choices uses the organization of the *decision* part of Dynaco (c.f. Section III).

The first simulator is used to study the behavior of the patterns by varying the workers workload, the time to process requests, the measurement and prediction errors of computation time, in function of various distributions of random numbers.

On top of the three presented patterns (one with and one without probes for DIET), an optimal pattern, without extra cost and without measurement and prediction errors, is implemented. It is used to calibrate the indices, such that, as written before, an index of 1 means a performance gap of 5% with an optimal pattern; the performance measure being relative to the QoS objective.

For each objective, simulations can be done by varying the value of one characteristic at a time. It enables to verify that the patterns behave as expected and to ponderate the characteristics by pattern or, if needed, to correct the behavioral functions.

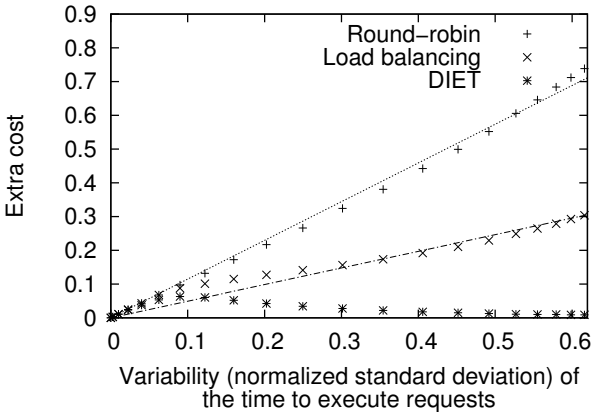


Fig. 5. Extra cost rate w.r.t. an optimal distribution of requests, due to the variability of the time to execute requests

For example, Figure 5 shows the extra cost due to the *variability of the time to execute requests* when the objective is to maximize the application's execution speed. It shows clearly that this extra cost can be approximated linearly for round-robin, by a function of equation $y = 1.15 \cdot x$. We can also notice that the extra cost tends to be linear for the load-balancing pattern ($y = 0.49 \cdot x$), except for small values where it is not efficient. For DIET, we notice that the prediction of the time to execute requests (here with an average prediction error of 8%) becomes efficient when the variability increase. This shows that we can use the function $y = 1.15/0.05 \cdot x$ for round-robin, $y = 0.49/0.05 \cdot x$ for load-balancing and $y = 0$ for DIET.

This simulator has been used to validate, refine or correct half of the nine characteristics. For the others, either they do not need this work since their behavior is well known, as for example "number of requests being processed" ; or the simulator describes not precisely enough the exact behavior of the pattern to be able to draw conclusions from it, which is the case for the pattern DIET and the characteristic "the number of requests being processed divided by the number of workers". This limitation can be lifted by doing complementary measurements of those used in Section IV

The second simulator is used to simulate which adaptation choices are done and when they are done, when the characteristics of the distributed system and the master-worker collection evolve with time. Thus it can ensure the consistency of the choices and their relevance. This simulator is to be used in concurrence with the first one to check if the choices done were the choices to be done. It can be used in a real test phase to plan interesting test scenarios.

For example, it was used to validate the choice to require to use a score gap of 5% between the current pattern and the replacing one before to exchange patterns. It was also used to compute the indices presented in Section V

VI. Conclusion

Today, it is still difficult to write applications for distributed systems. Thus, we chose to use the master-worker paradigm to get a high level abstraction of the system and to implement it in software components in order to adapt it. Then, we presented a framework to adapt the component-based master-worker collection. We then focused on an algorithm to choose when to change the master-worker pattern. To this end, we first characterized the distributed system and the master-worker collection, then we used this characterization to build the decision algorithm, then we briefly presented the simulations we used to validate it.

Through this paper, we described the process of designing the dynamic adaptation of the master-worker paradigm. Using the experimental plateform remains to be done in order to completely validate our algorithm. Then, it would be possible to use this design to build a framework for parametric and distributed applications which take into account the dynamic behavior of these applications and their execution environment.

In this study, we focused on the conception of algorithms to decide when to adapt and which pattern to choose. However, for the final conception of the global framework, it would be interesting to better study how to switch between patterns efficiently. It would also be appropriate to study the use of several masters or several worker collections inside the same application. In addition, it would be relevant to study ways to specify the QoS objectives more formally, to enable the user to have a better control over the dynamic adaptation.

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 the French National Agency for Research project LEGO (ANR-05-CIGC-11).

References

[1] D. Anderson, S. Bowyer, J. Cobb, W. S. D. Gbye, and D. Werthimer, "A new major SETI project based on Project SERENDIP data and 100,000 personal computers," in *Astronomical and Biochemical Origins and the Search for Life in the Universe*. Bologne, Italie: IAU Colloquium 161, 1997, p. 729.

[2] *Grid Computing and New Frontiers of High Performance Processing*. L. Grandinetti, Elsevier, 2005, ch. NetSolve: Grid Enabling Scientific Computing Environments.

[3] P. Combes, F. Lombard, M. Quinson, and F. Suter, "A Scalable Approach to Network Enabled Servers," in *Proceedings of the 7th Asian Computing Science Conference*, janvier 2002, pp. 110–124.

[4] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka, "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing," in *J. Grid Computing*, vol. 1, no. 1, 2003, pp. 41–51.

[5] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Ressource Management and Scheduling System in a Global Computational Grid," in *High-Performance Computing*, vol. 1, no. 1. Chine: IEEE CS Press, USA, 2000, p. 283.

[6] H. Bouziane, C. Pérez, and T. Priol, "Modeling and executing Master-Worker appplications in component models," in *11th Int. Workshop on HIPS*, Rhodes Island, Greece, avril 2006.

[7] C. Szyperski, *Component Software: Beyond Object Oriented programming*. Addison-Wesley Longman Publishing Co., Inc, 2002.

[8] F. André, H. L. Bouziane, J. Buisson, J.-L. Pazat, and C. Pérez, "Towards Dynamic Adaptibility Support for the Master-Worker Paradigm in Component Based Applications," in *Corregrid Symposium*, Rennes, France, août 2007.

[9] J. Buisson, F. André, and J.-L. Pazat, "Supporting adaptable applications in grid resource management systems," in *8th IEEE/ACM International Conference on Grid Computing*, Chicago, USA, 19-21 September 2007.

[10] H. L. Bouziane, "De l'abstraction des modèles de composants logiciels pour la programmation d'applications scientifiques distribuées," Ph.D. dissertation, École Doctorale MATISSE, février 2008.

I A chemical metaphor to model service selection for composition of services

A chemical metaphor to model service selection for composition of services

Claudia Di Napoli, Maurizio Giordano
Istituto di Cibernetica "E. Caianiello" - C.N.R.
Via Campi Flegrei 34, 80078 Pozzuoli, Naples - Italy
(c.dinapoli, m.giordano)@cib.na.cnr.it

Zsolt Németh
MTA SZTAKI Computer and Automation Research Institute.
P.O. Box 63, H-1518 - Hungary
zsnemeth@sztaki.hu

Nicola Tonellotto
Istituto di Scienze e Tecnologie dell'Informazione - C.N.R.
Via G. Moruzzi 1, 56124 Pisa - Italy
nicola.tonellotto@isti.cnr.it

Abstract

In the context of Internet of Services (IoS), Service Based Applications are composed of a number of possibly independent services that are available in a network and provided by many actors under different conditions (like price, time to deliver, and so on). Service provision conditions may change in time depending on provider policies, and as such they cannot be statically advertised together with the service description. In this paper we propose and investigate the possibility to use the chemical computational model [1] to address the problem of finding compositions of services that satisfy time constraints coming from the structure of an abstract workflow against the time availability associated to each service component.

1. Introduction

Nowadays, many enterprises using the Internet have fundamentally changed the way they do business, they develop new offerings that make the most of online channels. Many services such as, but not limited to, shopping, music listening, movie renting, multiplaying, are available over the Internet, changing our way of living. This change is seen by many experts [9] as a web-based economic revolution, where the Future Internet (FI) will embrace not just information and content but also services and real world objects. In this new web-based service economy everything becomes a service. This Internet of Services (IoS) relies largely on a service-oriented architecture (SOA), a flexible, standardized architecture that allows various applications to be combined into interoperable services. The enterprises take up the role of *service providers*, while the users exploit such services in combined ways to fulfill their needs. These Service Based Applications (SBAs) are composed of a number of possibly independent services, available in a network, which perform the desired functionalities.

In this context, it becomes necessary to organize compositions of services on demand in response to dynamic requirements and circumstances. It is likely that in service-oriented systems more service providers can provide the same functionality at different conditions, or even the same provider can provide one functionality at different conditions. These conditions refer to non-functional characteristics of a provided service (like price, time to deliver, and so on) that can change in time depending on both provider policies and consumer's needs. For example, the cost associated to a service may vary according to market conditions (driven by demand-supply mechanisms), or the time to deliver may vary according to the workload of the provider. Given the dynamic nature of service non-functional characteristics, they cannot be statically advertised together with the service description, but they have to be specified by the provider at the time a service is required.

The problem addressed in the present work is to select service instances that match the functional and non-functional requirements of an *abstract workflow*, i.e. a workflow that specifies only service functionality and their dependence constraints, to find one or more, if any, in-

stances of the abstract workflow. The non-functional requirements taken into account for the provision of a workflow are the time intervals [10, 13] during which each component can be potentially delivered. We call *service bid* the provision of a service in a fixed time interval. Only service instances whose time to deliver is compatible with the time constraints coming from the dependences occurring in the abstract workflow should be selected to provide the actual workflow. Of course, according to the number of service instances available for each component, and to the time to deliver of each instance, it will be possible to find zero or more workflows. As it can be conceived, with a large number of workflow activities and a large number of potential (dynamic) service offers, the problem can be of high complexity even without optimization efforts.

Nature metaphors are especially useful when it is difficult to formalize or compute an exact solution – typically in case of NP-hard problems, optimization, search problems, symbolic computation and so on. In many cases it is not even necessary to find the optimal solution just be reasonably close to it. By imitating nature processes, certain parameters are governed by nature laws that are known to evolve towards equilibrium, energy minimum, living entities may cooperate, share information, reproduce or die and so on. If appropriate links are set up between the nature processes and the computing problems, one may get a self-evolving, self-coordinating system where each individual steps are primitive and potentially unpredictable while the overall system evolves towards a known and expected state.

An early example for nature inspired problem solving is simulated annealing, first described by [12], [7]. In this scenario controlled cooling of melted metal is simulated numerically as crystals are formed: atoms can move randomly from their initial positions while lowering temperature; by random changes in position they can form crystals with lower energy level than initially. By carefully establishing relations to certain physical parameters in this model, e.g. temperature, energy, position, etc. known NP-hard optimization problems, like mapping[6], scheduling [14], routing [8] have been solved by this heuristics.

We propose to use the chemical metaphor [1] to represent the problem of selecting service instances in order to find workflows, if any available, that can be enacted to deliver a required application. In the chemical model of computation, chemical reactions involve molecules (i) that are within each others proximity (ii) and their actual conditions (chemical properties, temperature, energy, etc.) enable the reaction. In other words, the chemical matter transforms by local interactions. Each such interaction (their precise location, time and order) is a primitive and unpredictable step but their overall effect governs the matter into a predictable state.

Applying this metaphor to the problem of selecting service instances in order to obtain a workflow to be enacted allows to reduce the search space considerably. Moreover, by modeling the computation with chemical reactions means that the problem solution is implicitly modeled as an "evolving" system able to find more compositions when new bids are available in the system. In fact, in the chemical model of computation it is possible to change at runtime the state of the system (in this case the number of bids available), so allowing new solutions to be found (because new "reactions" may take place) in a way that simulate an adaptation of the system to different configurations that are not planned in advance.

This paper is aimed at establishing a baseline formal framework that will be used as a platform for modeling chemical optimization methods. These future concepts are outlined in the conclusions of the paper.

2. Problem formalization

It is assumed that SBAs are described in terms of user's requests specifying both the functionality of each component of the application, and the dependence constraints occurring among the components, i.e. the order of execution in which the components should be delivered. Such a user request is expressed in the form of an *abstract workflow* (AW) [11], i.e. a Directed Acyclic Graph (DAG) $AW = (S, E)$ where $S = \{S_i, \ldots, S_n\}$ is a set of nodes in the graph, and $E \subseteq S \times S$ is a set of directed edges in the graph.

Each node represents a required functionality, i.e. a service interface whose actual implementation can be provided by one or more services instances that can be provided with different non-functional characteristics. Each directed edge represents a data, or a control (or both) dependence between the pair of nodes it connects.

In the current formalization, abstract workflows include *chain*s and *branche*s. A chain is a path in the DAG whose nodes, except for the first and the last one, have just one incoming edge and one outgoing edge. The first node may have more incoming edges, but just one outgoing edge, while the last node may have more outgoing edges, but just one incoming edge. A branch is a set of more than one distinct paths that share only the first and the last nodes we refer to as respectively the branch *split* and *join* nodes.

Services whose non-functional characteristics match the dependence constraints coming from the whole applications have to be selected to obtain, from the abstract workflow, the actual workflow to be enacted. We refer to an actual workflow as an *instantiated workflow* (IW), or just workflow.

In the present work, the only non-functional characteristics taken into account for the provision of an actual work-

flow are the time intervals during which each component can be potentially delivered. Only service instances whose time to deliver is compatible with the time constraints coming from the dependences occurring in the abstract workflow should be selected to provide the actual workflow. Of course, according to the number of service instances available for each node in the graph, and to the time to deliver of each instance, it will be possible to find zero or more instantiated workflows.

3. The chemical computational model

Most algorithms are expressed sequentially even if they describe inherently parallel activities. Gamma (General Abstract Model for Multiset Manipulation) [4] aimed at relaxing the artificial sequentializing of algorithms. The basic data structure of Gamma is the multiset, i.e., a set that may contain multiple occurrences of the same element and that does not have any structure or hierarchy. Multisets are affected by so called reactions. The effect of a reaction (R, A), where R and A are closed functions, on multiset M is to replace in M a subset of elements $\{x_1, ...x_n\}$ such that $R(x_1, ...x_n)$ is true, by elements of $A(x_1, ...x_n)$ [4]. This model yields a multiset rewriting system where the program is represented by a set of declarative rules that are atomic, fire independently and potentially simultaneously, according to local and actual conditions. There is no concept of any centralized control, ordering, serialization, rather the computation is carried out in an indeterministic, self-evolving way. A Gamma program is inherently parallel. It has been shown in [4] that some fundamental problems of computer science (sorting, prime testing, string processing, graph algorithms, etc.) can be expressed in Gamma.

γ-calculus is a formal definition of the chemical paradigm from which all these chemical models can be derived. The fundamental data structure of the γ-calculus is the multiset M. γ-terms (molecules) are: variables x, γ-abstractions $\gamma\langle x\rangle.M$, multisets (M_1, M_2) and solutions $\langle M\rangle$ [2]. Note, that molecule is a synonym for all γ-terms. Juxtaposition of γ-terms is commutative ($M_1, M_2 \equiv M_2, M_1$) and associative ($M_1, (M_2, M_3) \equiv (M_1, M_2), M_3$). Commutativity and associativity are the properties that realize the 'Brownian-motion', i.e., the free distribution and unspecified reaction order among molecules that is a basic principle in the chemical paradigm [5].

γ-abstractions are the reactive molecules that can take other molecules or solutions and replace them by other ones by reduction. Due to the commutative and associative rules, the order of parameters is indifferent; molecules, solutions participating in the reaction are extracted by pattern matching – *any* of the matching ones may react. The semantics of a γ-reduction is $(\gamma\langle x\rangle.M), \langle N\rangle \rightarrow_\gamma M[x := N]$ i.e., the two reacting terms on the left hand side are replaced by the body of the γ-abstraction where each free occurrence of variable x is replaced by parameter N if N is inert, or x is hidden in M, i.e., it only occurs as a solution $\langle x\rangle$ [5]. Reactions may depend on certain conditions expressed as C in $\gamma\langle x\rangle\lfloor C\rfloor.M$ that can be reduced only if C evaluates to true before the reaction [2]. Reactions can capture multiple molecules in a single atomic step.

Besides the associativity and commutativity, reactions are governed by: (i) law of locality [5], i.e. if a reaction can occur, it will occur in the same way irrespectively to the environment; and (ii) membrane law [5], i.e. reactions can occur in nested solutions or in other words, solutions may contain sub-solutions separated by a membrane. The γ-calculus is a *higher order* model, where abstractions – just like any other molecules – can be passed as parameters or yielded as a result of a reduction.

The γ-calculus shows some similarities with the λ-calculus. Like the λ-calculus establishes the theoretical foundation for functional languages, the γ-calculus plays the same role for languages realizing the chemical paradigm. The Higher Order Chemical Language (HOCL) [3] is a language based on the Gamma principles more precisely, the γ-calculus extended with expressions, types, pairs, empty solutions and names. HOCL uses the self-explanatory **replace... by... if...** construct to express rules. **replace** P **by** M **if** C formally corresponds to $\gamma(P)\lfloor C\rfloor.M$ with a major difference: while γ-abstractions are destroyed by the reactions, HOCL rules are n-shot and remain in the solution nevertheless, single-shot γ-style rules can also be added. **replace... by... if...** is followed by **in** $\langle ...\rangle$ that specifies the solution the active molecule floats in. Notable features (extensions) of HOCL are: types,= that can be added to patterns for matching; pairs in form of $A_1 : A_2$ where A_1 and A_2 are atoms; and naming that allows to identify and hence, match rules, e.g.

$$\mathbf{let}\, inc = \mathbf{replace}\, x\, \mathbf{by}\, x + 1\, \mathbf{in}\, \langle 1, inc\rangle$$

that specifies an active molecule called inc which captures an integer and replaces it with its successor, floating in a solution together with an integer 1.

4. The chemical-based workflow

The core of the chemical model is a solution (that corresponds to a program) where molecules and other sub-solutions float (shortly: molecules unless there is a need for making difference.) Molecules may be active, representing the procedures or functions and passive, representing data. There are many ways to model a certain problem in the chemical paradigm; which entity acts as an active molecules and which entities are subjects of these actions

as passive ones. In this example we model all workflow components and service bids as passive molecules whereas the way how they are mapped and transformed are active molecules. All these form the chemical solution that coordinates the workflow composition.

An abstract workflow is represented by a set of *Node*s representing workflow service interfaces and *Edge*s representing control/data dependencies between services in the workflow.

A *Node* is defined as an embedded solution:

$$Node_i = \langle id : s_i, in : n_i, out : m_i, ... \rangle \tag{1}$$

where the attribute-value pairs have the following meaning: the attribute id is a unique integer identifying the workflow node, in is the number of incoming edges in the node (in-degree), and out is the number of outgoing edges (out-degree) from the node. Other attributes not relevant to the computation considered in the present work may follows.

A *Edge* is defined as another embedded solution:

$$Edge_l = \langle from : s_i, to : s_j, ... \rangle \tag{2}$$

where attribute id is a unique integer identifying the workflow edge, $from$ is the identifier of the node (source) originating the edge while to is the identifier of the node destination (sink) of the edge. Other, optional attributes may follow: for instance a pair $type : datadep$ denote the edge as a data dependence edge.

For example, in the chemical model the abstract workflow of Fig. 1 is represented by the set of solutions:

$$\begin{array}{l} Node_1 = \langle id : 1, in : 0, out : 2 \rangle, \\ Node_2 = \langle id : 2, in : 1, out : 1 \rangle, \\ Node_3 = \langle id : 3, in : 1, out : 1 \rangle, \\ Node_4 = \langle id : 4, in : 1, out : 2 \rangle, \\ Node_5 = \langle id : 5, in : 1, out : 1 \rangle, \\ Node_6 = \langle id : 6, in : 1, out : 1 \rangle, \\ Node_7 = \langle id : 7, in : 1, out : 1 \rangle, \\ Node_8 = \langle id : 8, in : 1, out : 1 \rangle, \\ Node_9 = \langle id : 9, in : 1, out : 1 \rangle, \\ Node_{10} = \langle id : 10, in : 2, out : 1 \rangle, \\ Node_{11} = \langle id : 11, in : 2, out : 0 \rangle \end{array} \tag{3}$$

note that node 0 and 11 are the *start* and *stop nodes* of the workflow since they have respectively no incoming and no outgoing edges. Nodes 1 and 4 are *split nodes* since they have more than one outgoing edges, while nodes 10 and 11 are *join nodes* as they ave more than one incoming edges.

In terms of node connectivity the abstract workflow of Fig. 1 is represented by the following edge solutions:

$$\begin{array}{l} Edge_1 = \langle from : 1, to : 2 \rangle, \\ Edge_2 = \langle from : 1, to : 3 \rangle, \\ Edge_3 = \langle from : 2, to : 4 \rangle, \\ Edge_4 = \langle from : 3, to : 5 \rangle, \\ Edge_5 = \langle from : 4, to : 6 \rangle, \\ Edge_6 = \langle from : 4, to : 7 \rangle, \\ Edge_7 = \langle from : 5, to : 8 \rangle, \\ Edge_8 = \langle from : 6, to : 9 \rangle, \\ Edge_9 = \langle from : 7, to : 10 \rangle, \\ Edge_{10} = \langle from : 9, to : 10 \rangle, \\ Edge_{11} = \langle from : 10, to : 11 \rangle, \\ Edge_{12} = \langle from : 8, to : 11 \rangle \end{array} \tag{4}$$

In Fig. 1 a chain is represented by the path: $Edge_4, Edge_7$, while the path $Edge_1, Edge_3$ is not a chain since the first node s_1 has more than one outgoing edge. A branch is represented by the two paths $Edge_5, Edge_8, Edge_{10}$ and $Edge_6, Edge_9$ since they share only s_4 as split node and s_{10} as join node.

For each service interface s_i a set of actual service implementations are available, $e^i_1, \ldots, e^i_{m_i}$. Each e^i can be delivered in a time interval, $\Delta t_{e^i} = T_{e^i} - t_{e^i}$, that means a service provider is willing to offer a service implementations by committing to provide its execution in the specified time interval. It is possible that a provider can offer more than one time interval for the service it provides and so it can provide more offers for the same service as soon as more time becomes available.

This offer, we call *bid*, and it is expressed in the chemical formalism by the solution:

$$\langle e^i : s_i, t_{e^i} : T_{e^i} \rangle \tag{5}$$

where the e^i is a generic endpoint of service i offered in the time interval $\Delta t_{e^i} = T_{e^i} - t_{e^i}$.

For example in the workflow of Fig. 1 the service endpoints bids are represented by the atoms:

$$\begin{array}{l} \ldots, \\ \langle e^3_1 : s_3, t_{e^3_1} : T_{e^3_1} \rangle, \\ \langle e^3_2 : s_3, t_{e^3_2} : T_{e^3_2} \rangle, \\ \langle e^3_3 : s_3, t_{e^3_3} : T_{e^3_3} \rangle, \\ \langle e^3_4 : s_3, t_{e^3_4} : T_{e^3_4} \rangle, \\ \ldots, \\ \langle e^6_1 : s_6, t_{e^6_1} : T_{e^6_1} \rangle, \\ \langle e^6_2 : s_6, t_{e^6_2} : T_{e^6_2} \rangle, \\ \ldots, \\ \langle e^8_1 : s_8, t_{e^8_1} : T_{e^8_1} \rangle, \\ \langle e^8_2 : s_8, t_{e^8_2} : T_{e^8_2} \rangle, \\ \ldots, \\ \langle e^9_1 : s_9, t_{e^9_1} : T_{e^9_1} \rangle, \\ \ldots \end{array} \tag{6}$$

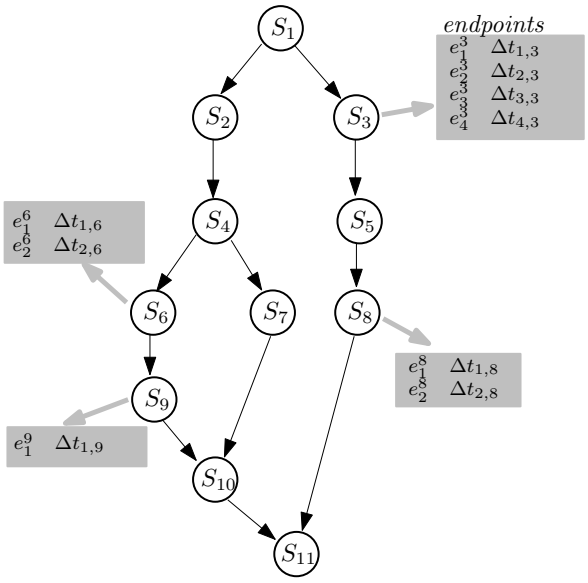


Figure 1. Abstract workflow and the associated bids

5. Selecting services using the chemical approach

As described in the previous section, the same service interface can be provided by one or more endpoints with different time intervals that can be associated to the same or different providers. In order to obtain the workflows that can be then enacted, a process to select the "right" services takes place. This process is expressed in terms of molecules reactions that occur when some conditions are satisfied. In the case considered in this work the conditions concern the compatibility of the time to deliver of each component with the dependence constraints specified in the abstract workflow.

Instantiated workflows are created from the abstract workflows by assigning certain service endpoints to their nodes. Thus, apart form the abstract workflow in Eq. 3, Eq. 4 and the offered services in Eq. 6, a third sort of solution is introduced, the instantiated workflow in the generic form of

$$\begin{aligned}&\langle start : \langle e^i : s_i\rangle, stop : \langle e^k : s_k\rangle,\\ &\quad node : \langle e^m : s_m\rangle,\\ &\quad node : \langle e^n : s_n\rangle,\\ &\quad \ldots,\\ &\quad t_{e^i} : T_{e^k}\rangle\end{aligned} \tag{7}$$

that represents a chain in the workflow where one end node is s_i associated with e^i, the other end node is s_k associated with e^k, there are some intermediate nodes (if any, s_m in this case) listed and the corresponding time frame is $t_{e^i} : T_{e^k}$.

5.1. Instantiating elementary chains

In order to select the suitable service implementations to obtain a workflow, first chains are considered. To this end a rule is defined so that bids associated to nodes belonging to a chain are linked together to form partial solutions of the complete workflow if the corresponding time intervals respect the dependence constraints (data/control dependences) and time constraints. In this step a link in the abstract workflow is turned into an instantiated workflow.

$$\begin{aligned}\textbf{replace}\quad &\langle id : s_i, in : n_i, out : m_i\rangle,\\ &\langle id : s_j, in : n_j, out : m_j\rangle,\\ &\langle from : s_i, to : s_j\rangle,\\ &\langle e^i : s_i, t_{e^i} : T_{e^i}\rangle,\\ &\langle e^j : s_j, t_{e^j} : T_{e^j}\rangle\\ \textbf{by}\quad &\langle id : s_i, in : n_i, out : m_i\rangle,\\ &\langle id : s_j, in : n_j, out : m_j\rangle,\\ &\langle from : s_i, to : s_j\rangle,\\ &\langle start : \langle e^i : s_i\rangle, stop : \langle e^j : s_j\rangle, t_{e^i} : T_{e^j}\rangle\\ \textbf{if}\quad &m_i = 1 \wedge n_j = 1 \wedge T_{e^i} \leq t_{e^j}\end{aligned} \tag{8}$$

The **replace** part of the rule matches five molecules: two nodes and an edge connecting them; and two service bids that match the specified interface.

The **if** part of the rule is the condition for applying the rule, that is a conjunction the following sub-conditions:

1. $(m_i = 1)$, that is s_i (and e_i as well) has exactly one

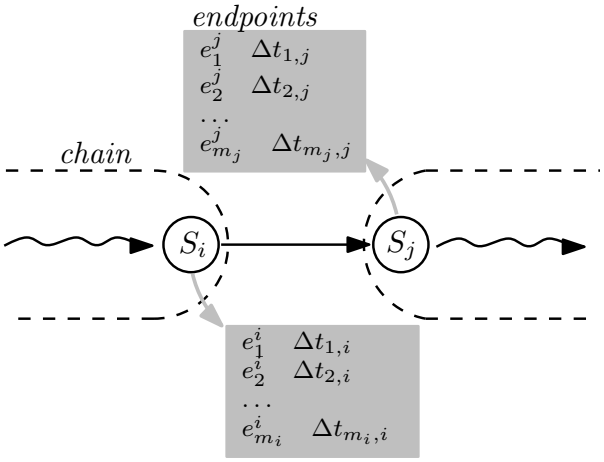


Figure 2. Composing chains

outgoing edge, i.e. it is neither the split ($m_i > 1$) node of a branch nor the terminal ($m_i = 0$) node.

2. ($n_j = 1$), that is s_j (and e_i as well) has exactly one incoming edge, i.e. it is neither the join ($n_j > 1$) of a branch nor the initial ($n_j = 0$) node.

3. ($T_{e^i} \leq t_{e^j}$), that is e_i and e_j are combined only if the time intervals have a null intersection, i.e. the time offers respect the execution order of the services s_i and s_j.

The **by** part of the rule is the resulted molecules. The two nodes and the edge are catalysts they are necessary to ignite the reaction but remain in the solution without any changes. Note, that the captured service bids are eliminated from the solution, i.e. they are already reserved by some nodes but the nodes remain in the solution thus, they may find other mappings if there are available and appropriate bids. These alternative bindings would be combined by chain and branch rules or selected by optimization methods.

Alternatively, it is also possible to define a rule for single node instantiation as:

$$\begin{array}{rl} \textbf{replace} & \langle id : s_i, in : n_i, out : m_i \rangle, \\ & \langle e^i : s_i, t_{e^i} : T_{e^i} \rangle, \\ \textbf{by} & \langle id : s_i, in : n_i, out : m_i \rangle, \\ & \langle start : \langle e^i : s_i \rangle, stop : \langle e^i : s_i \rangle, t_{e^i} : T_{e^i} \rangle \\ \textbf{if} & m_i = 1 \wedge n_i = 1 \end{array} \tag{9}$$

5.2. Combining chains

Instantiated chain fragments can be combined together by the following, named *chain rule*.

$$\begin{array}{rl} \textbf{replace} & \langle start : \langle e^l : s_l \rangle, stop : \langle e^i : s_i \rangle, t_{e^l} : T_{e^i}, \omega_1 \rangle, \\ & \langle start : \langle e^j : s_j \rangle, stop : \langle e^k : s_k \rangle, t_{e^j} : T_{e^k}, \omega_2 \rangle, \\ & \langle id : s_i, in : n_i, out : m_i \rangle, \\ & \langle id : s_j, in : n_j, out : m_j \rangle, \\ & \langle from : s_i, to : s_j \rangle, \\ \textbf{by} & \langle id : s_i, in : n_i, out : m_i \rangle, \\ & \langle id : s_j, in : n_j, out : m_j \rangle, \\ & \langle from : s_i, to : s_j \rangle, \\ & \langle start : \langle e^l : s_l \rangle, stop : \langle e^k : s_k \rangle, \\ & \quad node : \langle e^i : s_i \rangle, node : \langle e^j : s_j \rangle, \\ & \quad t_{e^l} : T_{e^k}, \omega_1, \omega_2 \rangle \\ \textbf{if} & m_i = 1 \wedge n_j = 1 \wedge T_{e^i} \leq t_{e^k} \end{array} \tag{10}$$

The **replace** part of the rule matches five molecules: two node atoms and an edge atom connecting them; the other two molecules represent already instantiated chains of bids that can be combined using he matched edge. The first molecule is the generic chain on the left of fig.2, while the second molecule is the right one in the same figure.

The **if** part of the rule is the same as that of Eq. 8.

The **by** part of the rule is the action. Note, that nodes and edges are catalysts, again. The resulted molecule is a chain that is a concatenation of the two captured ones. The captured chain fragments are transformed by the reaction and no longer are in the solution nevertheless, the rule in Eq. 8 may generate other instances of the same molecules yielding other possible combinations.

5.3. Condensing branches

The *branch rule* combines all the chains of bids that represent partial solutions in the graph, to form new molecules linked together by split and join nodes according to the ab-

stract workflow structure. Note, that this rule works simultaneously with the two chain rules: reactions are governed by the availability of the appropriate molecules.

The *branch rule* is:

$$
\begin{aligned}
\textbf{replace}\quad & \langle start : \langle e^{i_1} : s_{i_1}\rangle, stop : \langle e^{j_1} : s_{j_1}\rangle,\\
& \quad t_{e^{i_1}} : T_{e^{j_1}}, \omega_1\rangle,\\
& \langle start : \langle e^{i_2} : s_{i_2}\rangle, stop : \langle e^{j_2} : s_{j_2}\rangle,\\
& \quad t_{e^{i_2}} : T_{e^{j_2}}, \omega_2\rangle,\\
& \langle start : \langle e^{i} : s_{i}\rangle, stop : \langle e^{k} : s_{k}\rangle,\\
& \quad t_{e^{i}} : T_{e^{k}}, \omega_3\rangle,\\
& \langle start : \langle e^{l} : s_{l}\rangle, stop : \langle e^{j} : s_{j}\rangle,\\
& \quad t_{e^{l}} : T_{e^{j}}, \omega_4\rangle,\\
& \langle from : s_k, to : s_{i_1}\rangle,\\
& \langle from : s_k, to : s_{i_2}\rangle,\\
& \langle from : s_{j_2}, to : s_l\rangle,\\
& \langle from : s_{j_2}, to : s_l\rangle,\\
\textbf{by}\quad & \langle start : \langle e^{i} : s_{i}\rangle, stop : \langle e^{j} : s_{j}\rangle,\\
& \quad node : \langle e^{i_1} : s_{i_1}\rangle, node : \langle e^{j_1} : s_{j_1}\rangle,\\
& \quad node : \langle e^{i_2} : s_{i_2}\rangle, node : \langle e^{j_2} : s_{j_2}\rangle,\\
& \quad node : \langle e^{l} : s_{l}\rangle, node : \langle e^{j} : s_{j}\rangle,\\
& \quad t_{e^{i}} : T_{e^{j}}, \omega_1, \omega_2, \omega_3, \omega_4\rangle,\\
& \langle from : s_k, to : s_{i_1}\rangle,\\
& \langle from : s_k, to : s_{i_2}\rangle,\\
& \langle from : s_{j_2}, to : s_l\rangle,\\
& \langle from : s_{j_2}, to : s_l\rangle\\
\textbf{if}\quad & T_{e^k} < t_{i_1} \wedge T_{e^k} < t_{i_2} \wedge T_{j_1} < t_{e^l} \wedge T_{j_2} < t_{e^l}
\end{aligned}
\tag{11}
$$

The **replace** part of the rule specifies (see fig.3): four molecules representing chains to be connected, the two edge outgoing from the split node s_k, and the two edge incoming to the join node s_l. The four chain molecules are: the chain including the split node, the left and right chains (or atoms) representing the two branches originating from the split and converging to the join node, and the chain including as first node the join node of the branch, respectively.

The **if** part of the rule is the condition for applying the rule that checks if the end time of the service e_k precedes the start times of both the two branches, and if the start time of e_l follows both the end times of the two branches.

The **by** part of the rule is the action. The new molecule is the result of combining the two branch chains with the split and join nodes, more precisely with the chains the split and join nodes belong to. Like before, edge molecules are catalysts and remain in the solution for further reactions.

These rules constitute the core of the chemical composition mechanism. They can be considered as a 'single step' in a procedure where the series of such steps form the self-optimization process. How these steps are coordinated however, are future research work. The possibilities and directions are summarized in the next section.

6. Future Work and Conclusions

In this paper the chemical computation paradigm is used to model workflow composition for Service Based Applications according to constraints coming from the execution order of its components, and the time availability of the providers able to provide the actual implementation of these components.

The variability of the number of providers available to provide the services, and their availability in time (that can change for several reasons like the provider workload distribution, its market strategies in providing the service and so on), makes it necessary to rely on approaches that allow to find sub-optimal solutions based on some heuristics in a reasonable time and to compute new solutions every time conditions in the system changes.

Establishing a chemical model for workflow composition has three challenges. The first stage, baseline model has been introduced in this paper. It is aimed at finding an initial mapping between workflow activities and service bids. In this phase the chemical reactions act like constraint solvers: the carefully selected patterns and conditions can significantly reduce the potential reactions and hence, the search space. Further refinements can enhance complexity reduction and quality improvement to the initial mappings.

The second challenge is how established mapping can be improved. The environment is assumed to be highly dynamic where new bids may be added or existing bids may be changed and the workflow system must be able to adapt to the new conditions. If no changes occur in the chemical system, the chemical solutions are inert and they represent the calculated service compositions (results), if any. Inserting new bids into the system re-activates it allowing the computation of new workflow compositions until a stable state is reached again.

Differently from the standard approaches in the literature for this kind of problems (Linear Programming, Constraint Based Programming), the chemical approach behaves in an evolutionary way since partial solutions are not discarded and they can be reactivated to form new solutions when the workflow configuration changes. In this way, it is also possible that an existing mapping evolves into a more optimal one (according to some predefined criteria) with the availability of new bids. The baseline model consists only of constraints that enable or disable a reaction. This can be refined by adding further properties that make a reaction more likely or less likely, i.e. defining the chemical affinity of the molecules. Dimensions that make a reaction more likely can be the performance, reliability, availability of the bids whereas the tendency of forming molecules can be decreased by properties like price, waiting queue length or quality of offered bids just to mention a few examples. It is possible to model in the chemical paradigm

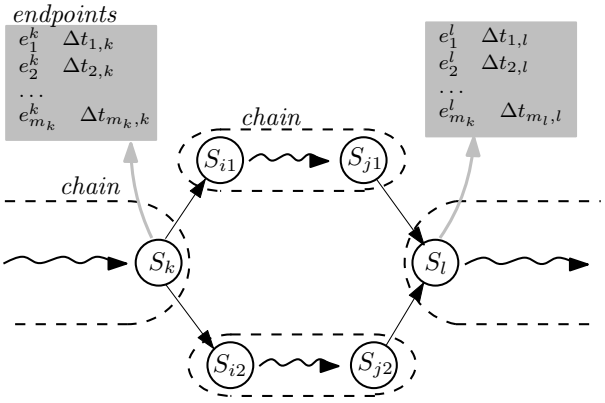


Figure 3. Condensing branches

that certain molecules split and rebound with some others. In such reactions it is evaluated whether the affinity of the free molecules are higher than that of the bound ones; in terms of optimization if the observed properties of the free molecule are better than that of the bound one. Hence, dynamic changes induced by added bids can change existing molecules whereas it is also possible that two bound molecules (mapped workflows) mutually split each other and then recombine from the yielded molecules in a more optimal way. The real research challenges in this phase are (i) which parameters can and should be observed and modeled as dimension of affinity; (ii) how the affinity functions can be established and (iii) what decision mechanism is necessary to evaluate the affinities, decide whether a certain reaction is possible under certain criteria for optimization.

Third, the temporal behavior of reactions must be controlled as well in order to reach a stable state in reasonable steps. The research challenge here is to find a mechanism that enables reactions without generating an unreasonably complex solution space. Such mechanisms may include chemical agents that neutralize, block or slow down processes; considering energy levels that may enable or disable certain reactions; or temperature that can affect the Brownian motion of the molecules and the likelihood of their reactions.

In the present work, the baseline chemical model is presented to establish a chemical framework where more sophisticated experiments for optimization can be realized.

7 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).

References

[1] J.-P. Banâtre, P. Fradet, and D. L. Métayer. Gamma and the chemical reaction model: Fifteen years after. In *WMP '00: Proceedings of the Workshop on Multiset Processing, LNCS 2235*, pages 17–44, London, UK, 2001. Springer-Verlag.

[2] J.-P. Banâtre, P. Fradet, and Y. Radenac. Principles of chemical programming. In *Fifth International Workshop on Rule-Based Programming, RULE'04, Electronic Notes in Theoretical Computer Science*, 2004.

[3] J.-P. Banâtre, P. Fradet, and Y. Radenac. Generalised multisets for chemical programming. *Math. Struct. in Comp. Science*, 16:557–580, 2006.

[4] J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Commun. ACM*, 36(1):98–111, 1993.

[5] J.-P. Banâtre, Y. Radenac, and P. Fradet. Chemical specification of autonomic systems. In *Proc. of the 13th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering (IASSE'04)*, 2004.

[6] P. Banerjee, M. Jones, and J. Sargent. Parallel simulated annealing algorithms for cell placement on hypercube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):91–106, 1990.

[7] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[8] Y. Cui, K. Xu, J. Wu, Z. Yu, and Y. Zhao. Multi-constrained routing based on simulated annealing. In *ICC '03. IEEE International Conference on Communications*, pages 1718–1722, 2003.

[9] DG Information Society and Media. Future internet 2020: Visions of an industry expert group, May 2009.

[10] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *In Proc. of the 11 th International Conference on Advanced Information Systems Engineering (CAiSE99), Springer Verlag, LNCS 1626*, pages 286–300. Springer-Verlag, 1999.

[11] D. Hollingsworth. Workflow handbook 1997. Technical Report TC00-1003, Workflow Management Coalition, 1995.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, 1983.

[13] O. Marjanovic. Dynamic verification of temporal constraints in production workflows. *Agile Development Conference/Australasian Database Conference*, 2000.

[14] A. YarKhan and J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing, LNCS 2536*, pages 232–242, London, UK, 2002. Springer-Verlag.

J Execution of Scientific Workflows based on an Advanced Chemical Coordination Model

Execution of Scientific Workflows based on an Advanced Chemical Coordination Model

Manuel Caeiro[1] and Zsolt Nemeth[2] and Thierry Priol[3]

[1] CoreGRID Network of Excellence / University of Vigo, E.T.S.E. Telecomunicación
E-36310 Campus Universitario, Vigo, Spain
[2] MTA SZTAKI Computer and Automation Research Institute, P.O. Box 63
H-1518 Budapest, Hungary
[3] INRIA, Campus Universitaire de Beaulieu
F-35042 Rennes Cedex, France

Abstract. Scientific workflows are being developed to support experimentation in an increased number of fields (e.g., astronomy, biology, nuclear physics, climate, geology). These workflows usually involve the management of large amounts of data and the realization of computational-intensive tasks, requiring the use of numerous computational resources (e.g., Grids). The work presented in this paper is based on the *modeling* aspect of artificial chemistries and aimed at establishing a highly abstract coordination model for distributed workflow enactment where decentralized control, autonomy, adaptation to dynamic changes, fault-tolerance, provenance, user steering are primary concerns. This paper introduces a new scientific workflow execution system based on a chemical coordination model where the notion of enactment is captured by a nature metaphor and envisioned as a chemical reaction that evolves autonomously according to local and actual conditions.

1 Introduction

A *workflow* is defined as a collection of coupled (computational) tasks intended to be processed in accordance with certain data-flow and control-flow prescriptions [1]. A *workflow execution system* is a computational system that performs the assignment of tasks to resources, controlling their execution and maintaining data-flow and control-flow dependencies.

In recent years, scientific workflows have been developed extensively [2] to support experimentation in an increased number of fields (e.g., astronomy, biology, nuclear physics, climate, geology). Distinguished features of these workflows are the management of large amounts of data and the realization of computational-intensive tasks, requiring the use of large amounts of resources (e.g., provided by Grids). Therefore, a main point in scientific workflow execution systems is to obtain reasonable performance from the numerous resources involved. Additionally, several other important requirements, e.g. dynamic changes, fault-tolerance, provenance, user steering must be taken into consideration in order to facilitate the realisation of scientific experiments in an appropriate way [3]. To exploit the

potential of full dynamicity, concurrency and autonomy, all these must be carried out at runtime and thus, enactment of scientific workflows requires a complex coordination between activities of the workflow and entities of the distributed system. Such a coordination comprises resource, service, data selection, handling control and data dependencies of activities, possibly fault detection and recovery in a large, highly dynamic, fault prone environment.

As in many cases, where the problem is complex and difficult to formalize, a nature metaphor may inspire a heuristic solution. To model workflow enactment we have considered a chemical analogy. Artificial chemistries are man-made systems that are similar to a real chemical system [4]. Dittrich et al. classify three main axes of applying artificial chemistry: modeling, information processing and optimisation. In this paper we explore the *modeling* aspect of an artificial chemistry based on multiset rewriting.

This paper introduces a new concept for workflow execution based on the chemical coordination model. This model is completely concurrent and enables to create programs without imposing any kind of sequentiality. Taking advantage of this concurrent solution the foundation logic of a workflow engine is described.

2 The Notion of Chemical Computing

Gamma was proposed in [5] to capture the intuition of computation as the global evolution of a collection of interacting elements (e.g., molecules). It can be introduced intuitively through the *chemical reaction* metaphor. Programs are conceived as *chemical solutions* involving a set of elements of different types that can react in accordance with certain *reaction conditions* and *actions*. Computation proceeds by replacing the elements satisfying the *reaction conditions* by the elements specified by the *actions*. The result of a *Gamma* computation is obtained when a stable state is reached, namely, when no more *reaction conditions* can take place. As an example, the computation of the maximum element of a non empty set can be described by the reaction rule "*replace x, y by x if* $x>y$". The chemical computation of this reaction finishes when a stable state is achieved and only the maximum element remains.

The *Gamma* formalism has been extended toward a higher-order model called γ-calculus [6]. In this model of computation every element (including reaction rules and actions) is considered as a molecule. Programs are made of *Active Molecules* (AMs), embedding reaction rules and actions, and *Passive Molecules* (PMs). AMs can capture AMs and/or PMs and produce new AMs and/or PMs according to reaction rules. Higher-Order Chemical Language (HOCL) is a language that evolved from the γ-calculus. HOCL enables to create computational programs as solutions of AMs, involving *reaction conditions* and *actions*, and PMs, representing data elements and resources.

3 A Workflow Execution System based on the Chemical Coordination Model

In the workflow definition phase an application to be executed is decomposed into a number of independent activities that are related to each other by data and control dependency, i.e. it is transformed into the so called *abstract workflow* that expresses the logic of the problem to be solved but it does not contain any specific means how to be executed. Subsequently, *abstract workflow* is transformed into a *concrete workflow* where each logical entity in the abstract workflow is assigned (in other terms: mapped, scheduled) to resource entities (i.e., resources, services, processes, etc.) that can enable the execution.

We envision workflow enactment, i.e. a procedure where an abstract workflow is mapped onto a concrete one dynamically and where the execution is controlled, similarly to chemical reactions. There are resources and workflow activities that may be matched in numerous combinations. Instead of picking one possible enactment pattern in advance, as any a priori schedule would do, workflow enactment in our model is a process that evolves in time. Properties of resources and activities define the possible matchings just like chemical properties define the affinity of molecules to react. Actual conditions enable a certain enactment step just like they define the potential of a reaction. If properties and conditions are well defined, the outcome of the coordination is predictable and controllable.

The entire chemical coordination takes place in a "chemical solution" where both workflow structure (activities) and resources are represented as chemical matter that at the same time define their possible interactions.

3.1 The Molecules

- *Task*. This molecule represents a task to be realized. *Tasks* may require some input *Data Elements* that have to be processed in order to produce some output *Data Elements*. In addition, *Tasks* need to be assigned to appropriate *Resources* for processing. In this way, *Tasks* may include information about the required features of the appropriate *Resources*.
- *Node*. This molecule is introduced to support the management of complex control- and data-flow prescriptions. *Nodes* can involve issues related to the control flow, such as splits and joins, or related to the data flow, such as the break down of a large data set into several smaller chunks of data. In this way, *Nodes* may involve the management and processing of *Data Elements*.
- *Activator*. This molecule is introduced to enable the control flow processing. The chemical paradigm is inherently concurrent and it does not impose any sequentiality to the computation. Nevertheless, workflow execution usually requires the management of sequences or other restrictions. In practice, *Tasks* and *Nodes* need to capture an appropriate *Activator* (or set of *Activators*) in order to be executed. In a similar way, they also generate new *Activators* to pass the execution control to the next *Tasks* and/or *Nodes*. They are quite

similar to *Tokens* in Petri Net models, but they are not used to maintain data elements.
- *Data Element*. This molecule represents a container of a piece of data used during the workflow execution. *Data Elements* are processed by *Tasks* and *Nodes* to obtain the intended outcomes and to manage the data-flow respectively.
- *Resource*. This molecule represents an entity that is able to perform the computation required by *Tasks*. Each *Resource* needs to include a description of its capabilities and features enabling the matching of appropriate *Resources* to *Tasks*.

3.2 The Execution States

Thus, the coordination is realized by the molecules and series of reactions in a chemical solution. The exact number and exact properties of all the molecules is assumed to be unknown. The solution may also contain control molecules that influence reactions – sometimes their role is similar to catalysts. The state of the computation is represented by the entire solution itself – it is a distributed information system by nature.

Within this solution *Tasks* and *Nodes* are AMs (Active Molecules) having specific *reaction conditions* and *actions*. They are able to capture other molecules. PMs (Passive Molecules) are *Activators*, *Data Elements* and *Resources*. Active molecules can have states (see Fig. 1):

- *Disabled Active Molecules* (DAMs). This state represents *Tasks* and *Nodes* that cannot be processed and are waiting for appropriate *Activators*. When a DAM reacts a matching *Activator* (or a set of *Activators*), a new *Enabled Active Molecule* (EAM) is resulted. The DAM remains in the solution (like a catalyst) and can generate a new EAM whenever it can react with appropriate *Activators*.
- *Enabled Active Molecules* (EAMs). This state represents AMs that do not have the appropriate input *Data Element* molecules. When an EAM finds appropriate *Data Element* molecules, the reaction produces a *Ready Active Molecule* (RAM) while the *Data Elements* remain in the solution.
- *Ready Active Molecules* (RAMs). This state represents a *Task* ready to be executed, but that has not been assigned to any resource. These molecules are waiting for an appropriate *Resource*. When a RAM finds such a *Resource* the reaction generates a *Initiated Active Molecule* (IAM).
- *Initiated Active Molecules* (IAMs). This state represents activities that are being executed by assigned *Resources*, carrying out the work described in the corresponding *Tasks*. When the execution finishes IAMs are reacted to generate *Resources*, *Activators* and *Data Elements*.

Figure 1 depicts states, state transitions and molecules. Notice that DAMs and *Data Elements* are represented with a broader line to show they are catalysts. Other molecules vanish in a reaction and new ones are generated.

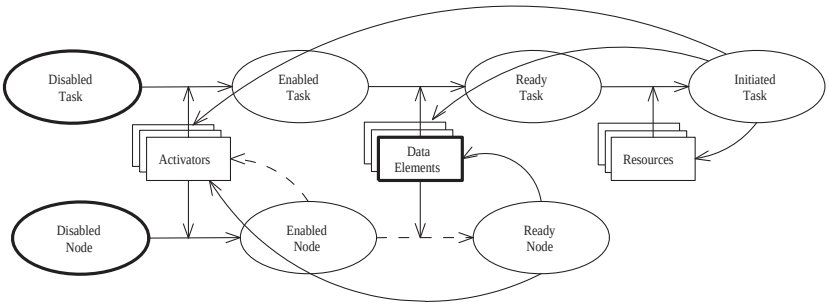


Fig. 1. State transitions for *Tasks* and *Nodes*

3.3 The Chemical Workflow Execution

This section introduces the whole proposal to support the workflow execution based on the chemical computation model. The abstract model of computation is depicted in Fig. 2. It is organized in four (separate) "chemical" sub-solutions for the sake of explanation. Each sub-solution is devoted to maintain the molecules that represent a certain state of execution. The molecules are transferred among sub-solutions through additional reactions not represented in the figure to simplify. The four sub-solutions contain molecules in accordance with the four states described in the previous section.

The behavior of the system can be interpreted following the numbers inside the circles in Fig. 2. It involves two main stages:

- The first stage involves the *compilation* of an abstract workflow description to produce the DAMs representing *Tasks* and *Nodes*. All the DAMs corresponding to the same workflow description include the same workflow identifier and version identifier. The workflow identifier is introduced to allow the execution of several workflows in the same chemical solution. Meanwhile, the version identifier is introduced to enable the execution of different versions of the same workflow, supporting the realization of dynamic changes to the workflow descriptions. Note, that the workflow descriptions could be in different workflow description languages. At this point, the key issue is to support the translation from the workflow descriptions to the chemical molecules.
- The second stage involves the *execution* of a workflow instance. Each instance corresponds with some input *Data Elements* that are introduced in the EAMs sub-solution. In addition, initial appropriate *Activators* have to be introduced in the DAMs sub-solution in order to begin the workflow execution. *Data Elements* and *Activators* carry appropriate workflow and version identifiers corresponding to the executed process. In addition, they carry an instance identifier in order to distinguish between different instances of the

same process. Notice that *Data Elements* and *Activators* can be introduced at any time.

After these two first states the workflow execution system evolves according to the chemical principles and the state transitions described in the previous section. Molecules can react if the three identifiers (workflow, version and instance) match. In addition, the generated molecules, representing the different states of *Tasks* and *Nodes* and the produced *Data Elements* and *Activators*, inherit the same identifiers. This feature enables the execution of different workflow processes and different instances of the same workflow in the same execution engine.

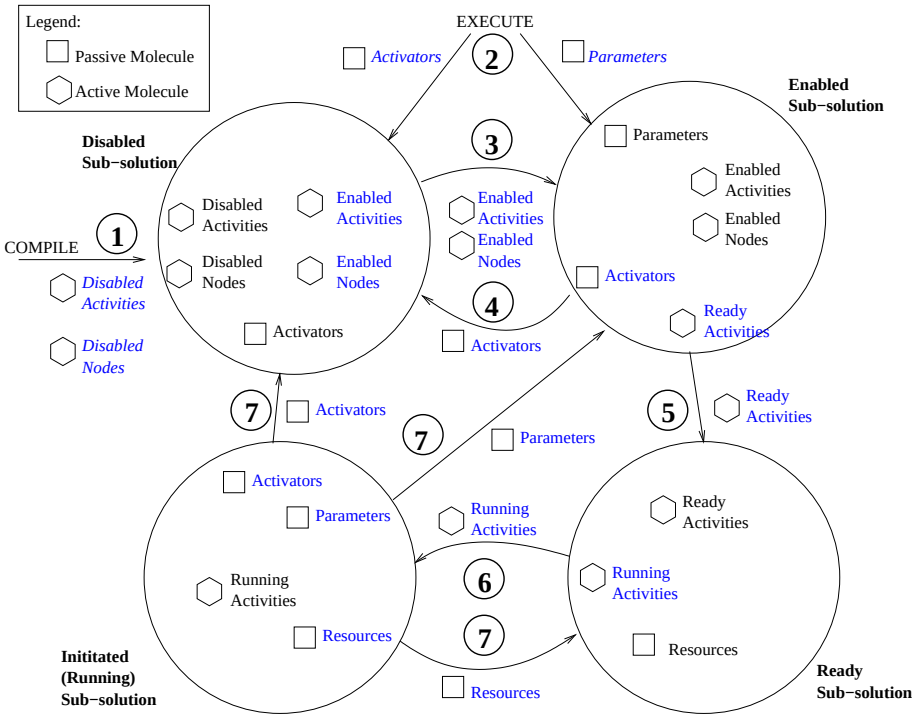


Fig. 2. A chemical abstract model of computation to execute scientific workflows

3.4 Node Types to Support Control- and Data-flow

Node molecules are introduced to support complex control- and data-flow structures in the execution of scientific workflows. Similarly to business workflows, scientific workflows involve the common types of behaviors described by the well-known workflow patterns [7]. In addition, there are specific data operators such

as the SCULF *one-to-one* and *all-to-all* that deserve attention [8]. The processing of these constructs can be appropriately performed in a chemical computation model by using specific *Nodes* types. Next, some of the more relevant types are described:

- *AND-Split* and *AND-Join*. They represent points where several tasks have to be initiated and several threads have to be synchronized before the next task can be performed, respectively. *AND-Split* looks for an input *Activator* and produces as much output *Activators* as required by the AND operator. *AND-Join* waits for the appropriate input *Activators* and produces one output *Activator* that enables next task.
- *OR-Split* and *OR-Join*. *OR-Split* represents a point where one or more tasks can be initiated. This connector waits for an input *Activator* and it produces one or more of several output *Activators*, based on the evaluation of conditions associated with each of the outgoing branches. This requires the creation of a corresponding *Node* EAM that has to be processed in the EAM sub-solution as it requires the checking and evaluation of some specific *Data Elements*. *OR-Join* involves an extra difficulty. In accordance with the pattern description [7], the branch following the *OR-join* receives the thread of control when either (i) each active incoming branch has been enabled in a given case or (ii) it is not possible that any branch that has not yet been enabled in a given case will be enabled at any future time. Hence, *OR-Split* should generate an additional specific *Activator* to be captured by the *OR-join* indicating the number of branches that have been effectively activated.
- Data *one-to-one*. This operator requires to combine the element produced by two sources in a one to one fashion. This connector will generate a set of corresponding *Node* EAMs as soon as it detects an appropriate input *Activator*. Each generated *Node* EAM will group a specific couple of input *Data Elements* into a pair. It is a requirement that input *Data Elements* need to be ordered. Each *Node* EAM will be related with elements at a specific position, namely: pair 1, pair 2, pair 3, etc. The initial *Activator* needs to carry the number of pairs of input *Data Elements* to be processed. If this number is not known beforehand and the elements are produced by the sources dynamically then new *Activators* have to be generated correspondingly, indicating the number of elements generated at each time.
- Data *all-to-all*. This operator requires to combine the parameters produced by several sources among them in all the possible combinations. In this case, every time an (or a set of) input *Data Element* is produced a new *Activator* has to be issued. The *Node* DAM should maintain an account of the number of *Activators* produced from each input source. Each time it receives a new *Activator*, it generates an appropriate number of *Node* EAMs of this type that have to look for appropriate *Data Elements* to compose the appropriate couples. Each *Node* EAM will look for a certain couple of input parameters and will generate such couple as soon as it finds them.

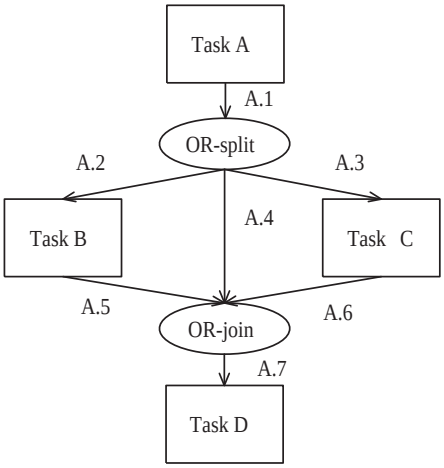


Fig. 3. Representation of an *OR-Split* and *OR-Join* construct with *Tasks*, *Nodes* and *Activators*. The *OR-Split* node generates the *Activator* A.4 that informs to the *OR-Join* node about the number of output *Activators* produced in the *OR-Split*.

4 Support of Dynamicity

Dynamicity is a main requirement in scientific workflows. Many times scientific work evolves through the processing of different versions of the same experiment, varying the task to be performed or the range of parameters to be used. The chemical based execution system is focused on supporting these needs to a long extent. Particularly, it is important to notice the following points:

- The distinction between *Activators* and *Data Elements* separates the management of the control flow from the data to be processed. This separation enables to conceive a steering facility through the unique management of *Activators*.
- *Data Elements* are never eliminated from the chemical solution. Both input *Data Elements* and produced *Data Elements* are maintained in the solution in order to enable the development of monitoring and provenance solutions, that can capture such elements in order to provide appropriate information about the execution. In addition, the maintenance of *Data Elements* facilitates the realization of re-runs from a specific point in the workflow process.
- New experiments can be initiated dynamically in parallel with the existing ones. As DAMs representing *Tasks* and *Nodes* remain in the solution and it is possible to maintain several active versions of the same process. Similarly, it is possible to maintain several active instances of the same process.

5 Related work

During the last years, several scientific workflow execution systems have been developed: Kepler [3], Triana [9], Taverna [10], etc. These systems are usually proposed as a proof of concept as they are oriented to support concrete applications focusing on particular problems. They usually include a typical workflow execution engine that takes the responsibility for the complete processing of the workflow execution and the management of dependencies. In most cases, their primary goals are neither concerned with the support of dynamicity nor of implicit parallelism; they are programmed as other workflow engines, adopting more-or-less sequential solutions.

More related with the solution proposed in this paper, there have been numerous attempts to develop workflow engines based on the concept of Event-Condition-Actions (ECA) rules. These attempts involve the management of a set of rules that control the triggering of certain events and the satisfaction of certain conditions, to activate new events or perform certain actions. Examples are the Petri Net [11] and Event-driven Process Chains (EPCs) [12] formalisms. Several workflow engines have adopted this approach in their design: [13], [14], [15], [16]. The main difference between our proposal and these ones is that we consider rules as first class entities that can be changed and modified. In these systems, rules are statically programmed into more or less sequential programs. Meanwhile, in our solution, rules are chemical molecules evolving naturally in parallel.

6 Conclusions

This paper introduces a new workflow execution system based on the chemical computation model. It has been shown that the γ-calculus can address all aspects of workflow modeling and enactment [17]. In the advanced model, presented in this paper, the notion of connectors is introduced in order to realize more sophisticated workflow constructs, the data and control flow aspects are separated and the states of the activities are introduced for better control of the execution. Furthermore, the chemical coordination model allow capture and modification of the active molecules themselves that enable a more dynamic and flexible representation of the workflows.

The chemical model is not an out-of-box solution for workflow enactment; many issues of workflow scheduling are not addressed directly. Rather, the model is an abstract framework where various advanced enactment strategies can be specified. As artificial chemistries have the modeling, information processing and optimization aspects, in this work we explored the modeling potential of the chemical metaphor with a future outlook for information processing and optimization.

Acknowledgments. The work presented in this paper is partially supported by CoreGRID Network of Excellence IST-2002-004265 and European Community's Seventh Framework Programme under grant agreement 215483 (S-Cube).

References

1. Fahringer, T., Qin, J., Hainzer, S.: Specification of grid workflow applications with agwl: an abstract grid workflow language. In: CCGRID '05: Proc. of the Fifth IEEE Int. Symp. on Cluster Computing and the Grid, Washington, DC, USA, IEEE Computer Society (2005) 676–685
2. Taylor, I., Deelman, E., Gannon, D., Shields, M.: Workflows for e-Science. Springer-Verlag (2007)
3. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and Computation : Practice & Experience **18**(10) (2006) 1039–1065
4. Dittrich, P., Ziegler, H., Banzhaf, W.: Artificial chemistries - a review. Artificial Life **7**(3) (2006) 225–275
5. Banâtre, J.P., Métayer, D.L.: Programming by multiset transformation. Commun. ACM **36**(1) (1993) 98–111
6. Banâtre, J.P., Fradet, P., Radenac, Y.: Programming self-organizing systems with the higher-order chemical language. International Journal of Unconventional Computing **3**(3) (2007) 161–177
7. Aalst, W.M.P.V.D., Hofstede, A.H.M.T., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1) (2003) 5–51
8. Glatard, T., Montagnat, J., Lingrand, D., Pennec, X.: Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. International Journal of High Performance Computing and Applications (2007)
9. Taylor, I., Wang, I., Shields, M.S., Majithia, S.: Distributed computing with triana on the grid. Concurrency - Practice and Experience **17**(9) (2005) 1197–1214
10. Oinn, T., Justin, M.A., Ferris, Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics Journal. **online** (June 16, 2004)
11. van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. Information & Software Technology **41**(10) (1999) 639–650
12. Keller, G., Nttgens, M., Scheer, A.W.: Semantische prozemodellierung auf der grundlage "ereignisgesteuerter prozeketten (epk)". Arbeitsbericht Heft 89, Institut fr Wirtschaftsinformatik Universitt Saarbrcken (1992)
13. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Deriving active rules for workflow enactment. In: Database and Expert Systems Applications. (1996) 94–115
14. Bae, J., Bae, H., Kang, S.H., Kim, Y.: Automatic control of workflow processes using eca rules. IEEE Trans. on Knowl. and Data Eng. **16**(8) (2004) 1010–1023
15. Zhang, G., Jiang, C., Sha, J., Sun, P.: Autonomic workflow management in the grid. In: International Conference on Computational Science (3). (2007) 220–227
16. Fjellheim, T., Milliner, S., Dumas, M., Vayssière, J.: A process-based methodology for designing event-based mobile composite applications. Data Knowl. Eng. **61**(1) (2007) 6–22
17. Németh, Z., Pérez, C., Priol, T.: Distributed workflow coordination: Molecules and reactions. In: The 9th International Workshop on Nature Inspired Distributed Computing, IEEE (2006) 241