FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

ViePEP – Vienna Platform for Elastic Processes

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Philipp Hoenisch
Matrikelnummer 0725710

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: o.Univ.-Prof. Dipl.-Ing. Mag. Dr. Schahram Dustdar
Mitwirkung: Dr.-Ing. Stefan Schulte
Dr. Srikumar Venugopal

Wien, 29. Januar 2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

ViePEP – Vienna Platform for Elastic Processes

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Philipp Hoenisch
Registration Number 0725710

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: o.Univ.-Prof. Dipl.-Ing. Mag. Dr. Schahram Dustdar
Assistance: Dr.-Ing. Stefan Schulte
Dr. Srikumar Venugopal

Vienna, 29. Januar 2013

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Philipp Hoenisch
Brunner Gasse 56-58, 2380 Perchtoldsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Acknowledgements

Since no one will ever read this... Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt felis ac erat ultrices fringilla. Phasellus tincidunt libero ac ante pulvinar semper. Suspendisse pulvinar pulvinar massa. Nullam quam nisl, viverra in ultrices vel, mollis sed dui. Aenean lobortis aliquet ligula nec accumsan. Vestibulum pharetra, nulla in varius mattis, orci ante pharetra lectus, vel elementum urna risus ultrices lorem.

Anyway, if you get that far, I would like to say a big thank to my supervisor at the Vienna University of Technology, he gave me a lot of feedback to my work and motivated me to improve myself from the first day on. Another special thank goes to Srikumar Venugopal from the University of New South Wales, without him this work would not be possible, it was a great honor to spend 4 great month at the UNSW where I have learned a lot about research and had overall a great time.

Abstract

Within this thesis, we propose a novel Business Process Management System for the Cloud. It is able to process several hundred workflows simultaneously while monitoring their executions in order to counteract against potential Service Level Agreement violations through scheduling the queued workflows and acquiring additional computing resources when needed or releasing unneeded ones.

Elastic Processes are a novel paradigm from the field of Cloud computing. It combines the various facets of elasticity that captures process dynamics in the Cloud. Elastic Processes are described in three ways: cost elasticity, resource elasticity and quality elasticity. Nowadays, these elasticities are also relevant for workflows in the Cloud. A workflow consists out of several individual processes, each requires a different amount of resources, follows different Quality-of-Service attributes and produces a different amount of costs. Since in today's workflows resource intensive tasks get more and more common, and the amount of workflows executed in parallel varies over time, the amount of needed resources will also diversify enormous. That is way it is necessary to acquire additional computing resources during the system's runtime, or release some whenever they are not needed anymore.

Therefore, we developed ViePEP – the Vienna Platform for Elastic Processes. ViePEP is on the one hand a Business Process Management System for the Cloud, capable to manage and process the execution of several hundred workflows simultaneously. It further monitors their executions in order to identify potential Service Level Agreement violations in time. And on the other hand, ViePEP is able to counteract against the lack of needed or the excess of used computing resources. Using a prediction model, ViePEP is not just able to counteract in-time but also predict the future need of resources for the near future in order to acquire additional resources punctual or release unneeded resources. By evaluating ViePEP with different configured experiments we have shown, that ViePEP is able to automatically process workflows, while taking care of Service Level Agreements through scheduling their executions and acquiring or releasing computing resources.

Kurzfassung

In dieser Arbeit stellen wir ein Business Process Management System (BPMS) für die Cloud vor. Dieses BPMS kann mehere hundert Geschäftsprozesse (BPs) simultan bearbeiten und ausführen. Diese BPs werden während der Ausührung beobachtet um auf etwaige Dienstgütervereinbarungen verstößße frühzeitig zu reagieren, mehr Resourcen anzuforder wenn nötig, oder nicht mehr gebrauchte Resourcen freizugeben.

Elastische Prozesse (EPs) sind ein neuartiges Konzept aus dem Bereich von Cloud Computing bei dem die Vielartigkeit der Elastizität der Cloud zum Einsatz kommt. EPs zeichnen sich durch 3 Eigenschaften aus: Kosten Elastizität, Resourcen Elastizität und Qualiät Elastizität. Diese Elastizitäten spielen heutzutage ebenfalls bei BPMSs in der Cloud eine große Rolle. Ein BP besteht aus mehreren einzelnen Prozessen die alle unterschiedliche Anforderungen an Resourcen haben, andere Qualiätskriterien gelten und somit auch unterschiedlich hohe Kosten anfallen. Die Anzahl der gleichzeitig auszuführenden BPs kann über die Zeit enorm variieren, und dadurch auch die Anzahl der benötigten Resourcen. Daher kann es notwendig sein, dass während der Laufzeit zusätliche Resourcen angefordert, oder nicht mehr benötigte Resourcen wieder freigegeben werden.

Wir entwickelten daher ViePEP – the Vienna Platform for Elastic Processes. ViePEP ist gleichzeitig ein BPMS welches in der Lage ist, mehrere hundert BPs gleichzeitig ausführen zu können, ihre Abarbeitungen zu beobachten und etwaiigen Dienstgüterverinbarungen verstößen frühzeitig entgegen zu steueren. Weiters ist es in der Lage BPs nach ihrer Priorität zu sortieren, so dass ausgemachte Vereinbarungen über die Dienstgüter der einzelnen Services nicht gebrochen werden. Weiters ist ViePEP in der Lage gegen etwaige Resourcenknappheit frühzeitig entgegen zu wirken. Mit Hilfe eines Prediction Models, kann ViePEP die benötigten Resourcen für die nahe Zukunft vorhersagen um zusätlich benötigte Resourcen rechtzeitig anzufordern oder unnötige freizugeben.Mittels ausführlichen Experimenten mit unterschiedlichen Einstellungen haben wir gezeigt, dass ViePEP in der Lage ist, automatisiert BPs auszuführen, auf alle Dienstgütervereinbarungen zu achten und automatisch neue Resourcen anzufordern oder freizugeben.

Contents

CHAPTER 1

Introduction

The following gives a short introduction to this thesis. Prefaced by the motivation for the topic follows the aim of the work. Afterwards, we give a overview about the thesis' structure.

1.1 Motivation

Business Process Management (BPM) is a multidisciplinary approach which covers organizational, management, and technical aspects, and "includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational Business Processs (BPs)" [70]. One specific subtopic of BPM is the automatic processing of BPs, which needs to be supported by concepts, methodologies and frameworks from the field of computer science [46]. Ludäscher et al. [41] defines the automated part of a business process as a business workflow. In many cases, (Web) services are composed to a dynamic business workflow that may span several organizations and computing platforms [56]. Each service in this workflow is also called business process or workflow step. Nowadays, resource-intensive tasks may not only be found within Scientific Workflows (SWF), but appear more and more often in BPs. So are compute and data-intensive analytical processes often found in the finance industry and systems for managing smart girds in the energy industry. In the latter one, data from a very large number of sensors has to be gathered automatically, preprocess, processed, analyzed and stored in order to offer customers consumption reports or even guarantee grid stability [59]. Since a business process consists out of several individual, dependent or independent process steps, it is necessary to define theirs functional requirements [18]. Beside of the functional requirements coming with the services, a number of non-functional requirements are added to such a business process description. An example is the timeliness of tasks, i.e., while some of these processes have to be carried out in real-time, others can be postponed to the future but still need to consider a particular deadline. Further, since the amount of process instances, which have to be carried out simultaneously, may vary to a very large extend, it is a difficult task to estimate the ever-changing demand of computing resources.

Therefore, a technology is needed which is able react to the dynamic change of needed computing resources while still ensuring the faultless BP execution, i.e., whenever additional resources are needed, the technology has provide them in any way, such that, the BP execution will not crash or stuck at a critical moment. Throughout the last couple of years, software engineering research and practice have put remarkable focus on the field of Cloud computing which is capable to do exactly this.

The definitions of Cloud computing are many-faceted, Buyya et al. [11] propose the following definition: "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on Service Level Agreements (SLAs) established through negotiation between the service provider and consumers." Another famous definition of Cloud computing has been introduced by National Institute of Standards and Technology (NIST) [43] "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". They further define five essential characteristics as : *On-demand self-service* – Cloud-based resources are provided on-demand to the end-users by request. *Broad network access* – the Cloud-based resources can be accessed worldwide using standard network mechanism. *Resource pooling* – the access to Cloud-based resources is based on a multi-tenant way. *Rapid elasticity* – the underlaying Cloud infrastructure allows dynamically scaling up and scaling down, i.e., by adding or removing computing resources. *Measured service* – the consumption of resources is measured and serves as a foundation for elastic pricing and billing models [43].

The relatively new technology of Cloud computing enables developers to realize so called *Elastic Processs (EPs)* [17]. The concept of these processes are defined by [17] “as the various facets of elasticity that capture process dynamics in Cloud and human computing”. The main properties for the economic and physical dynamics of EPs are *resource elasticity*, 2) *cost elasticity*, and 3) *quality elasticity*. The promise of Cloud computing enables developers to achieve software applications and frameworks considering all three ways of elasticity at the same time.

Therefore, a Business Process Management System (BPMS) is needed, which is able to carry out, several BPs simultaneously while still meeting their non-functional requirements, monitoring the executions in order to learn from the monitored data, thus, it can predict the future need of computing resources.

1.2 Aim of the Work

Within this thesis, we want to introduce a novel BPMS for the Cloud. It should be able to execute hundreds of BPs simultaneously, respond to unexpected or expected changes of the current needed resources in a way that it acquires further or removes unneeded Virtual Machines (VMs) during process runtime. Further, it will monitor the BP executions in order ensure the faultless BP executions and counteract against SLA violations in-time. In order to know, when the system needs additional resources, we will deploy an elastic reasoning algorithm, which is able to learn from the monitored data by applying a prediction model, thus it can forecast the future need of resources for each single service. Further, by charging a Load Balancer with balancing the invocation on all available resources, the framework ensures an optimal resource utilization.

In order to evaluate our approach, we present a prototypical implementation of an BPMS for the Cloud. We call this BPMS ViePEP – the Vienna Platform for Elastic Processes. ViePEP will be evaluated by running several different configured scenarios.

1.3 Organization

The remainder of this thesis is structured as following:

- Chapter 2 details the current State of the Art in the area of Cloud computing and Business Process Management. This chapter explains important terms in this area and presents some well approved standards.

- Chapter 3 provides an overview of related work in the area of service optimization, resource management, and BPM in the Cloud.
- In Chapter 4 we firstly present ViePEP – a Business Process Management system for the Cloud. We introduce a real world use case scenario in order to show how and where the results of this thesis can be applied. Afterwards, we define the single tasks which ViePEP is taking care of, i.e., Business Process Management and their executions, monitoring, and resource optimization.
- Chapter 5 covers a presentation of our prototype implementation of ViePEP. Further, the single components acting in ViePEP are explained. In addition, we give a detail description of our resource prediction approach.
- Chapter 6 shows how we evaluated the ViePEP while presenting and discussing the results of the experiments.
- Chapter 7 concludes this thesis with a short summary and gives an outlook of our future work.

CHAPTER 2

State of the Art

In this section, we give some background knowledge for this work. We discuss the underlying concepts of Service-Oriented Architectures, Cloud computing including Resource Provisioning, Business Process Management and Service Compositions, Quality-of-Service (QoS) attributes and how to monitor them. Furthermore, we discuss some basic methods for resource optimization.

2.1 Service-Oriented Architecture

Since software engineering may be a very complex, time consuming and expensive process, reusable software elements, provided as services can benefit vendors and as well customers. To achieve sustainable reusability, on a business-internal as well as on a Business-to-Business (B2B) level, a high degree of interoperability and integration is crucial. The concept of Service-Oriented Computing (SOC), [27,55] utilizes services as fundamental elements for developing applications. According to Papazoglou, services are "*self describing*, *platform-agnostic computational elements* that support *rapid*, *low-cost composition* of *distributed applications*" [55]. They perform functions, which can be anything from a simple requests to a complex business processes (BP). Erl writes in [20] that the important characteristics of services are:

- Loose coupling: Services are autonomous and not hard-wired. The relationship between services minimizes dependencies and allows for the replacement of single elements.

- Service contract: Services retain to a communications and interface agreement, as defined by one or more service description documents (e.g., SLA).
- Autonomy: Services have the single control over their implemented logic.
- Abstraction: A service is described by an interface in a service contract but they hide the actual implementation logic from the outside world.
- Reusability: The functionality provided by a service is aimed for reuse.
- Composability: Services can be assembled to form service compositions (see Section 2.3). In these compositions, the abstraction principle still applies, that means that to the outside world, composite services may not be distinguishable from an atomic service.
- Statelessness: Services do not retain an internal state. Context information is carried in the message exchange with the service and service caller.
- Discoverability: Services are designed to be describable so that they can be found via an accessible discovery mechanisms, e.g., Universal Description, Discovery and Integration (UDDI) [65].

In addition to these characteristics, the view of services is the basis for creating a Service-Oriented Architecture (SOA). Today's SOAs comprises much more than just services, service consumers and a registry (as depicted in the outdated SOA-triangle [45]). The SOA-triangle covers as well as message mediators, service buses, monitors, management and governance systems, workflow engines, and many other component types [56].

The SOA-triangle can be found in Figure 2.1 and consists out of three roles [48]:

- Service Provider: The *Service Provider* is the host who provides the service implementation. It *publishes* the *Service Description* to a *Registry*. The *Service Description* contains information about the service like what it is good for, what are the necessary input parameters, and how does the result look like.
- Registry: The *Registry* is the mediator between the *Service Provider* and the *Service Consumer*. It services as an database where Service Providers can register their (Web) services so that, Service Consumers can find and further invoke them. An example is UDDI, a service registry which is described in [65]

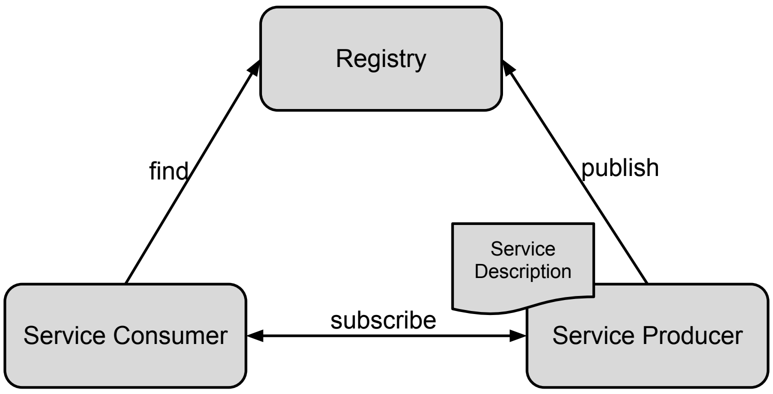


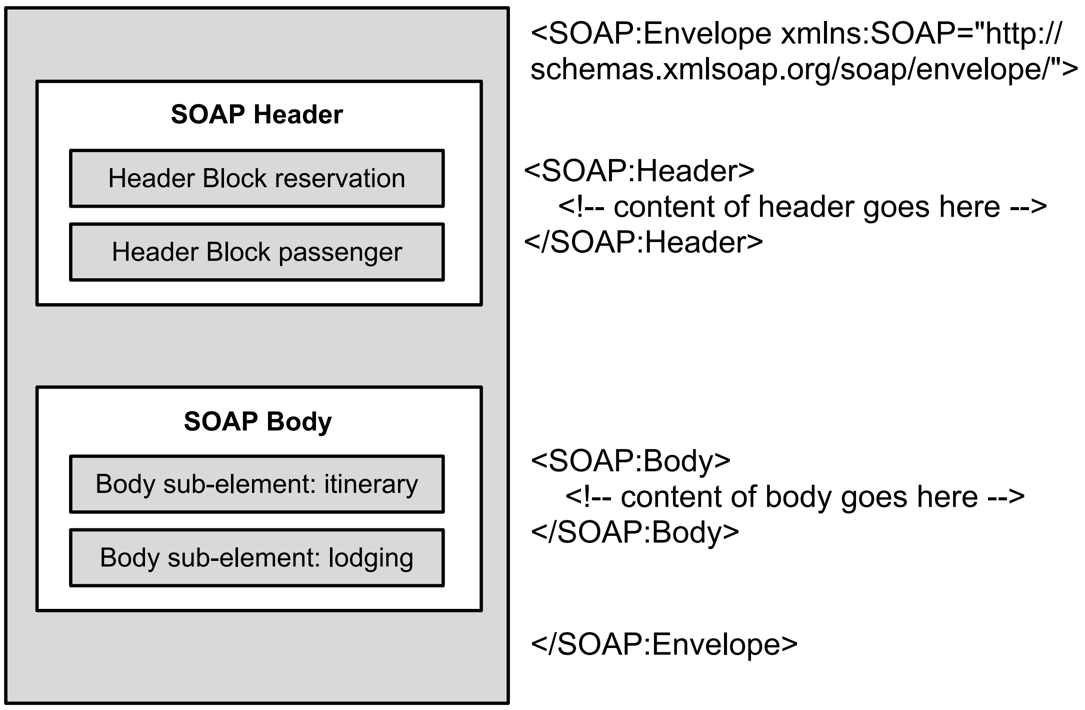
Figure 2.1: SOA-Triangle (adapted from [26])

- Service Consumer: The *Service Consumer* queries the *Service Registry* for a specific service. When an appropriate service was found, the binding takes place and the *Service Consumer* can subscribe to the wanted service in order to invoke it.

2.2 Web Services

A way to implement SOAs is to use Web service technologies [16,48,55]. Web Services are software components which are made available through the Internet or Intranet. As shown in Figure 2.1, Service Providers can publish their Web Services (using the Service Description). In addition to that, Web services can be discovered, subscribed and further invoked by Service Consumers. In order to interchange data Web services make use of the Extensible Markup Language (XML) [75]. According to Curbera et al., a Web Service framework is divided into three areas [16]:

- **Communication: SOAP**
  To allow Web services to be distributed and heterogeneous, a communication mechanism is needed which is platform independent, unified, secure and, in order to reduce network traffic, as light-weight as possible. XML [75] is nowadays a fully established standard for information and data encoding for platform-independence and internationalization. Therefore, building a communication protocol for Web Services using XML is a logical consequence. SOAP, originally defined as Simple Object Access Protocol was initially created by Microsoft and



```
<SOAP:Envelope xmlns:SOAP="http://
schemas.xmlsoap.org/soap/envelope/">

<SOAP:Header>
   <!-- content of header goes here -->
</SOAP:Header>

<SOAP:Body>
   <!-- content of body goes here -->
</SOAP:Body>

</SOAP:Envelope>
```

Figure 2.2: SOAP Envelope (adopted from [16, 76])

later implemented in collaboration with several companies including Developmentor, IBM, Lotus, and UserLand. Simple Object Access Protocol (SOAP) defines an XML-based protocol for messaging and Remote Procedure Calls (RPCs). It does not describe a new transport protocol, but works on top of existing ones such as HTML and SMTP. The core of a SOAP message is called the SOAP Envelope. It has a very simple XML structure consisting out of two child elements: one for the header and the other one for the body. Figure 2.2 shows a graphical representation of a SOAP Envelope and the XML format for it.

- **Description: WSDL**
  In the case of Web services, SOAP offers the basic communication, but it does not describe what messages must be exchanged to successfully invoke services. To solve this problem, WSDL (Web Service Description Language) was created. WSDL is an XML-based format which was developed by IBM and Microsoft to describe Web services as collections of communication endpoints that can ex-

change certain messages. In other words, a WSDL document describes a Web service interface and provides the users with a point of contact. At the time of writing this thesis, WSDL 2.0 is the current recommendation by W3C and has replaced its predecessor WSDL 1.1 [8]. Its conceptual model can be found in Figure 2.3. The Web service description contains the following parts:

  - Description: The root element *description* wraps the abstract and concrete parts of the service definition.
  - Abstract Part: The abstract part includes the service interfaces (this replaced the *portType* from WSDL 1.1 [77]), corresponding operations, input and output messages.
  - Types: The message types, also often called *parameters*, are needed to define the accepted types for the input and output parameters. They can consist out of subtrees in order to define simple and complex types.
  - Concrete Part: The concrete part contains the service binding, service endpoint (which replaces the *service port* from WSDL 1.1 [77]) and the actual service [8].

- **Discovery and Registry: UDDI**
  In order to provide Web services to several users a discovery mechanism is needed. UDDI [65] is an OASIS specification for Web service registry implementations. It defines a data model to describe service providers, what services they offer and the relations within the entities. Firstly, UDDI registries describe services verbally (informal) and secondly, it describes the services technically (formal) so that service consumers can use special query operations on order to look for services. Technical service descriptions usually link to the service WSDL definitions.

2.3 Business Processes

At the beginning of this thesis, we described briefly the purpose of the framework we want to create. We mentioned that a platform for Business Process Management shall be developed. BPM is a multidisciplinary approach, covering organizational, management and technical aspects and "includes methods, techniques, and tools to support the design, enactment, management and analysis of operational business processes" [70].

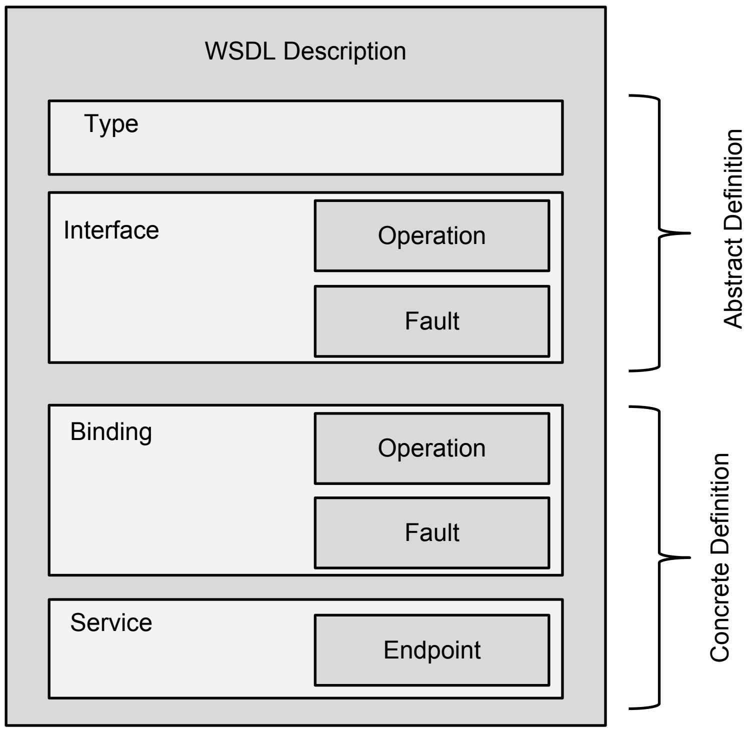


Figure 2.3: WSDL 2.0 (adapted from [61])

One particular subtopic of BPM is the automatic execution of modeled processes, which needs to be supported by concepts, methodologies and framework from the field of computer science [46]. The automated part of a business process is also known as a business workflow [41]. The notion *processes* is more specific than *workflows*. Processes can apply for physical or biological processes for instance. Anyway, processes may be distinguished from a workflow by the fact that a process has a well-defined input, output and purpose, while the notion of workflow may apply more generally to any systematic pattern of activity.

The basic infrastructure of Web services (described in Section 2.2), is sufficient to create application only involving single invocations. In contrast, we speak of a service composition (or workflow), which is a sequence of concatenated services, if a service's business logic involves the invocation of several Web services in a sequence. More detailed, sometimes it is necessary to provide a solution to a problem where a single service call is not sufficient to provide some functionality but a set of service calls are. In this case we speak about service compositions. Figure 2.4 shows an example workflow for booking a flight.

[18] gives an overview of several service composition approaches including *static* and *dynamic*, *automated* and *manual*, *business rule driven*, *model driven*, *context-based* and *declarative composition*. The most common approach for Web service composition is WS-BPEL which is presented in the following.

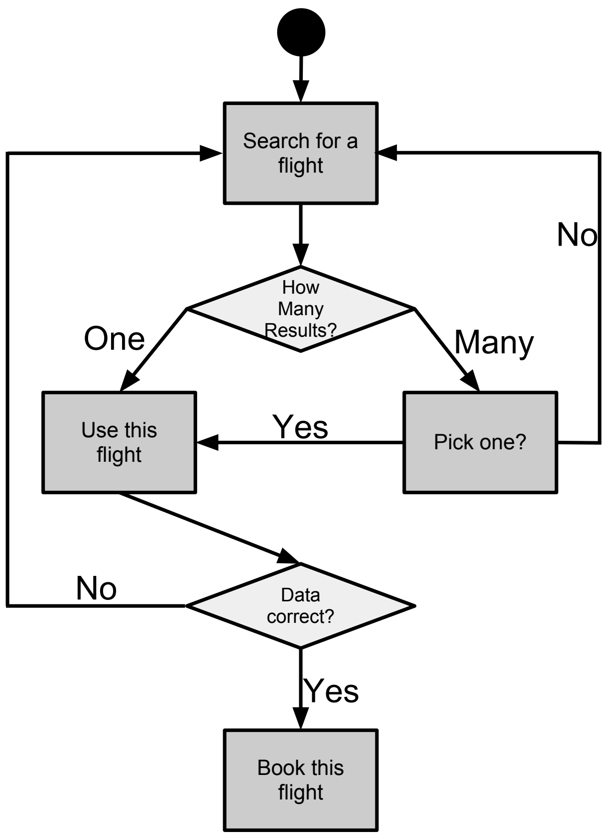


Figure 2.4: Workflow for Booking a Flight

2.3.1 WS-BPEL

The industry has reached a general agreement for a single orchestration specification: the OASIS Web Service Business Process Execution Language (WS-BPEL) [50]. WS-BPEL is an XML-based language, the single process steps are performed by Web services. It assumes that each service is defined using a WSDL file and policy assertions that identify any extended features. The participating Web services are specified by partnerLink definitions, which point to the respective portType declarations and service EndpointReferences (EPRs). If the WSDL portType definition of a service is meant to be an equivalent of a class interface definition in object-oriented programming, the partnerLink element can be seen as the equivalent object.

Since BPEL documents are very verbose, they are usually generated by tools having a graphical user interface where the main directives can be added in a drag-and-drop manner (e.g., Eclipse BPEL Editor [19], Netbeans BPEL Designer [47], Oracle BPEL Process Manager [54]). The finished BPEL document is parsed by a WS-BPEL parser engine. which publishes the process WSDL file and accepts incoming SOAP messages which requests the execution of this process. Upon receipt of a request message, the engine starts to interpret the XML-encoded directives given in the process definition. The main WS-BPEL directives are defined by OASIS [50]:

- **subscribe:** This instruction subscribes to an operation of another Web service. Input and output of the invocation are defined by one variable each.
- **assign:** WS-BPEL allows for the use of variables which can be assigned values using the assign instruction. The values can be of atomic nature (e.g., String, Integer,...) or an XML markup. In order to select the part of a variable which shall be used, XPath expressions [78] are used.
- **receive:** The receive instruction signifies that a message has to be received from an outside participant. This can be used either at the beginning of the process to let a user define a start parameter and to start the execution or in the middle of the process to receive results from asynchronous invocations.
- **while:** specifies a repetitive execution of some process steps.
- **switch:** conditional execution of parts of the process.
- **flow:** defines a parallel execution of several process steps.
- **reply:** returns the process result.

Finally, it is important to mention that BPEL has been designed for use with more than one target service and is not intended for intra-service protocol specification. Furthermore, the WS-BPEL specification does not (yet) define any human interactions: it is not possible for an user to *interrupt* a Business Process in order to manipulate variables or change its workflow. Since in many cases a human interaction in BPs is useful, OASIS defined a specification for a BPEL4PEOPLE, e.g., “This specification introduces a BPEL extension to address human interactions in BPEL as a first class citizen. It defines a new type of basic activity which uses human tasks as an implementation, and allows specifying tasks local to a process or use tasks defined outside of the process definition.” [49].

2.3.2 Service Level Agreement

While a WSDL file describes a Web service in a more technically manner (e.g., what methods are provided or what is the expected result, ...), a SLA can be seen as a contract between two services or between a service provider and a service consumer [11, 28, 60, 73]. A SLA guarantees specific levels of performance and reliability at a certain cost. “Usually, cost is the underlying factor that drives QoS that can be offered” as mentioned in [73]. This can easily explained, because as more money a user is willing to pay, the more resources are offered by the service provider.

A complete SLA is a legal document stating the parties which are involved, the terms of the agreement, application and the support services included. SLAs also specify what service providers can expect from their customers in terms of workload and resource usage. In addition to what has to be monitored and what the terms of this agreement are, the SLA document specifies what penalties are applied whenever a SLA was violated by a party.

Guarantees in a SLA are defined as a set of Service Level Objectives (SLOs) [9]. A SLO is a combination of one or more component measurements to which constraints are applied. A SLO is said to be in compliance if the underlying measurement values are within the specified constraints. SLOs may have operating periods during which the SLO has to be compliant. Because of the statistical nature of the Internet, it is usually not possible to offer guarantees that can always be met. Thus, a compliance period and a compliance percentage may be associated with SLAs. The compliance percent defines the percentage of time the SLA has to be compliant over the compliance period.

A SLA can contain SLOs for any kind of services, however, in this paper we want to focus on Web services. In total, there is no standard for SLAs, but IBM has defined a specification for Web Service Level Agreement (WSLA) [42]. This specification defines a SLA template for Web Services. The WSLA language is XML-based and is defined by an XML Schema [79]. It can be used by both, the service provider and the service customer.

An very important aspect of WSLA is its capability to deal with specification of particular domains and technologies. Therefore, the WSLA language was created to be extensible to include specific types of operation descriptions, (e.g., using WSDL to describe a Web services operation), measurement directive types for specific systems, special functions to compose aggregate metrics and predicates to evaluate specific metrics. Some of the most commonly defined SLOs include:

- **Service Provisioning:** This guarantees that the service will be provisioned in a certain way. An example is that a provider will provide the customer with redundant connection to its Web services.

- **Reliability:** Reliability metrics consist of availability guarantees over a period of time. Some examples are: no more than 1 hour of unscheduled downtime during the year for the network; the Web server will be available 99% of the time it is accessed over a period of a year.

- **Response time:** This metric defines the maximum time a service is allowed to take to respond to user requests. Example of this metric is 95% of users will have a response time of 3 seconds or less during the working hours, where working hours are between 8 am and 6 pm

- **Throughput:** This metric defines the number of Web service requests successfully served in a given time period. It is an important metric since service providers want to know how many users can be served within a given time period simultaneously without losing quality.

- **Finish before:** This metric defines a fix deadline. The service provider has to make sure that the wanted service invocation has happened before the defined deadline passed. An example is a fix date like 2012-12-24;18:00:00.

- **Costs:** The cost metric defines the maximal costs a customer is willing to pay. This metric is often connected to the “Finish before” metric and it is becoming an important metric because of the pay-as-you-go models, where you just pay for the resources you have acquired.

2.4 Cloud Computing

Cloud Computing [5, 10, 12, 24, 58, 71] is a new paradigm for delivering resources on demand to customers similar to other utilities like water, electricity or gas. Although there are many Cloud computing providers (e.g., Amazon EC2[1], Google App Engine[2], Microsoft Azure[3], Heroku[4], etc.) there is no standard taxonomy for it. Everyone tries to define Cloud computing and its services in their own way. However, the NIST [43] defines Cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” Further, they say that every Cloud-based resource is defined by five essential characteristics [43]:

- They can be automatically provided on-demand, when they are requested by a cloud consumer (Scale-out: horizontally scaling in the sense of adding additional resources [44]).
- They can accessed worldwide using standard network mechanisms.
- They support access in a multi-tenant way.
- Their consumption is measured and serves as a foundation for elastic pricing and billing models.
- The underlying Cloud infrastructure is able dynamic scale up and down in order to adapt to the changing demand of resources (Scale-up: vertically scaling in the sense of adding resources to a single node, e.q., adding additional CPUs or memory to a machine [44]).

Therefore, what everyone agrees on is that in Cloud computing everything can be seen as a service (XaaS). Most often there is talk of Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS), Hardware-as-a-Service (HaaS), etc. (see Section 2.4.2)

[1]`http://aws.amazon.com/ec2/`
[2]`https://appengine.google.com/`
[3]`http://www.windowsazure.com/`
[4]`http://heroku.com`

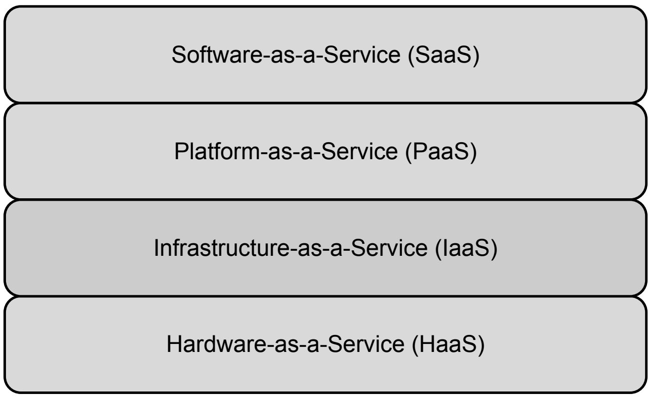

Software-as-a-Service (SaaS)
Platform-as-a-Service (PaaS)
Infrastructure-as-a-Service (IaaS)
Hardware-as-a-Service (HaaS)

Figure 2.5: Cloud Layered Architecture (adopted from [58])

2.4.1 Cloud Layered Architecture

The Cloud Layered Architecture is the design of different components layered on several levels. Each layer is defined by a service which supports the five characteristics specified by NIST [43] (shown in Figure 2.5). “Cloud architectures are underlying on infrastructure which is used only when it is needed that draw the necessary resources on-demand and perform a specific job, then relinquish the unneeded resources and often dispose them after the job is done.” [58]. The lowest layer forms the HaaS, it provides the needed hardware for IaaS. On top of it is a PaaS deployed which enables providers to deploy their software on top of it, this layer is the so called SaaS. All of these services are accessible from all over the world. Through Cloud providers such as Amazon, Google, Microsoft, ... it appears that that there is only one single point of access to these resources. There are three different kinds of Cloud models mentioned by [4]:

- **Public Cloud:** Public Cloud describes the Cloud mode in the traditional mainstream sense, where resources are provided on-demand over the Internet via Web services or Web applications.

- **Private Cloud:** This Cloud mode is managed within an organization and it is not limited by the restrictions of network, bandwidth, security, security exposures and legal requirements that may come with Public Clouds.

- **Hybrid Cloud:** The Hybrid Cloud is a mix out of the two other Cloud models.

2.4.2 X-as-a-Service

As mentioned before, in correlation with Cloud computing, everything can be seen as a service. There are many possible variations of XaaS like HaaS, IaaS, PaaS, SaaS, Communication-as-a-Service (CaaS), ... [23]. For the purpose of this thesis, it is sufficient to focus on the following [43]:

- **HaaS:** The Hardware-as-a-Service model is aimed to be an advantage for business users. They can rent the hardware in the sense of computing resources, and do not need to invest in building and managing data centers.

- **IaaS:** IaaS is the delivery of computing IaaS to customers. I.e., it is the idea of providing computing resources in the form of Virtual Machine to the customers. Beside of the higher flexibility, a key feature of IaaS is the usage-based payment models (pay-as-you-go model). This allows the customers to only pay what they use and grow/shrink in needed resources whenever they need to. Examples for IaaS providers are Amazon EC2[5], Windows Azure Virtual Machine (VM)s[6], Google Compute Engine[7], HP Cloud[8], ...

- **PaaS:** A Platform-as-a-Service is a category of Cloud computing services that provides a computing platform and a stack of solutions as a service. The big idea behind of PaaS is to provide a platform for developers with all the system libraries and environments so that they can control the deployment and configuration of their software. The purpose of PaaS is to facilitate the deployment of applications without the cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities. A popular example for an PaaS provider is the Google App Engine[9].

- **SaaS:** As HaaS, IaaS and PaaS, Software-as-a-Service can as well be seen as a multi tenant platform. It uses common resources and a single instance of both, the object code of an application as well as the underlying database in order to serve multiple customers simultaneously. SaaS is deployed on top of the Platform

[5]`http://aws.amazon.com/ec2/`
[6]`http://www.windowsazure.com/`
[7]`https://cloud.google.com/products/compute-engine`
[8]`https://www.hpcloud.com/`
[9]`https://appengine.google.com/`

as a Service which provides interfaces to its underlying layers and it is commonly referred to the Application Service Provider (ASP) model.

2.4.3 Virtualization

Virtualization is a key feature of Cloud computing. This concept abstracts the coupling between the operating system and the hardware. In order to improve agility, flexibility and to reduce costs and thus to enhance business values, it refers to the abstraction of the logical resources away from the underlaying physical resources. Basically, virtualizations in Cloud computing are of different types: server virtualization, where a software layer is added to a real machine in order to support a desired architecture in form of VM [62], storage virtualization and network virtualization. A common interpretation of server virtualization is the mapping of physical available resources to multiple virtual (logical) representations or partitions. In such an environment it is easy to create new virtual computing environments, expand, shrink or move as demand varies [1].

2.4.4 Fault Tolerance

Since Cloud computing relies by nature on complex system, i.e., the splitting of software components into several pieces and distribute them on different layers, the requirement for fault-tolerance to achieve reliability has become a critical issue [25,64]. By using the key feature of Cloud computing, namely virtualization, Cloud providers and software developers, are enabled build more fault-tolerant systems. This fault-tolerance can be achieved on different levels, i.e., on VM level it is possible to create a back-up copy of each VM instance, so that, in case of an error in an instance, the affected instance can be restored by creating a new instance out of the back-up. Similar applies to the SaaS layer, whenever a software service is not reachable anymore, it can easily be redeployed on a different VM. However, the challenges in this area multifaceted, it is desired to prevent errors and faults in advent, that is why many researchers are focusing on complex data mining and machine learning models in order to be able to predict and prevent faults.

2.4.5 Interoperability

Interoperability is needed to allow applications to be ported from one cloud to another, or to have one application deployed on several cloud instances, e.g., for higher performance. The recently created Cloud Computing Interoperability Forum (CCIF)[10] was formed to define an organization that would enable interoperable enterprise-class Cloud computing platforms through application integration and stakeholder cooperation. However, this is very optimistic and it turned out that vendor lock-in is a big problem in today's world of Cloud computing. Vendor lock-in, is the situation in which the customers are dependent on a single supplier for a product (i.e., a good or service), or products, and cannot move to another vendor without substantial costs and/or inconvenience. This topic was addressed by Frank Leymann et al. in [7]. In his successive paper [6] he proposed the framework TOSCA to realize portable Cloud services. Armbrust et al. [5] propose using multiple providers at the same time in order to address the problem of vendor lock-in. This approach is often defined as the Cloud federation [35]. Cloud federation can be used within the service model layer (horizontal federation) or across layers (vertical federation). Also, it can be permanent (redundancy) or temporary (migration) nature.

2.5 Elastic Processes

The authors of [17] proposes the phrase *EP* in connection processes and services.

Cloud computing and human computing have features in common like *dynamic resource requirements and provision* and *QoS within processes*. However, this *elasticity* captures the biggest essence of Cloud computing: when limited resources are offered for potentially unlimited use, providers must manage them elastically by scaling up and down, as needed. But, as it is common today, understanding and supporting elasticity purely from the view of resource management, which is rather restrictive. Resources' requirements are not determined only by the application using them. Therefore [17] proposes the definition of *Elastic Processes* in three ways:

- **Resource Elasticity:** Describes the elasticity in the sense of the acquired or used resources. Since the demand of resources (e.g., CPU, RAM, network bandwidth) may vary during runtime, it is necessary to react to this need. Therefore it might be essential to acquire more resources in order to expand or release unneeded resources to shrink.

[10] `http://www.cloudforum.org/`

- **Cost Elasticity:** Cost elasticity describes the resource provision's responsiveness to changes in cost. Service providers apply this cost elasticity when they want to define a price model for their Cloud computing system. In this context, the cost elasticity is referred to as utility computing, in which resources such as computational services provided by VMs, network bandwidth, and storage services provided on different storage hierarchies are charged-based on so-called pay-as-you-go pricing mechanism. For example Amazon and other IaaS providers define usually for their services at least two pricing models: *On-demand instances* are a pure pay-as-you-go price model in which the customers do not have a longterm commitment. And, *Spot instances* occur when spot prices fluctuate over time according to supply-demand status and other factors. Users bid the maximum price they are willing to pay for these instances and run them as long as the spot price is lower than the bidding price, or until the instance is explicitly terminated. The service provider can charge higher prices on peak times and lower prices during off-peak time using these *Spot prices* in order to shape customers behavior and that more flexible users would tend to consume rather during off-peak times and than during peak times.

- **Quality Elasticity:** Measures how responsive the QoS is to the change of used resources. This elasticity comes from the idea, that the more resources are consumed, the better the achievable quality is. To achieve this, and underlaying algorithm is needed, that can decide whether more resources are needed to improve the quality or some can be freed. The main challenge here is to define a measurable quality and cost function in order to compute the resource requirements for a given quality attribute, such as maximal execution time. However, in most cases some kind of reasoning is required which does not give an exact answers, but an approximate or quick result.

To summarize it, EPs combine the concepts of Business Processes (and their automated execution) with the idea and the possibilities of Cloud computing. However, there are many challenges and problems coming with this idea which are described in more detail in Chapter 3.

2.6 Resource Optimization

The abilities of Cloud computing facilitates flexible and efficient resource management via virtualization at anytime and from anywhere, so that customers can get the demanded IT resources easily. In this environment, the servers are shared by different applications. Furthermore, the Cloud resources are offered in distinct types of VMs. The needed resources may not be static and the permanent provisioning of computing resources which are able to handle peak system loads is obviously not the best solution, as the capacities will not be utilized most of the time. Therefore, a lot of research has happened in this area: [81] proposes an approach that combines resource consumption prediction and resources allocation in the Cloud that provides VMs to users. This approach allocates resources while the VM is starting for load balancing, named Statistic-based Load Balance (SLB). SLB includes online statistical analysis of VM's performance and resource demand forecasting and a load balancing algorithm choosing a proper host. [29] proposes a way using agents which are assigned to single resources, using this methology the provided resources can be used efficiently.

Out of this, resource optimization can be divided in several fields including: (but not limited to): load balancing, reasoning about the needed resources, demanding of additional resources, releasing of unneeded resources, move/copy/delete services from instances, and many more. In many cases, load balancing in combination with resource optimization using data mining or reasoning is used, which are described in following:

- **Load Balancing:** The goal of load balancing [32], is to improve the performance by balancing the load among various resources (e.g., CPUs, discs, network links, VMs, ...) to achieve an optimal resource utilization, maximize the throughput and minimize the response time by avoiding an overloaded system.
- **Resource Optimization:** Optimizing a Cloud system can be done in two different ways: firstly, reactive, i.e., whenever the load is getting in a critical state, a new resource is acquired and the load is balanced, or secondly, proactive, i.e., the needed resources are demanded before the system can get in a critical state, this is often done using a machine learning approach [74]. In both cases, a monitoring component has to record the used resources.

Since resource optimization is a big challenge in Cloud computing, it is a hot research topic in computer science and many researchers all over the world take up that challenge. We present relevant research results in the next Chapter and consider them as related work for this thesis.

CHAPTER 3

Related Work

In this chapter related work in the field of this thesis will be presented. The presented approaches are compared on several criteria that can be found in Table 3.1. The table consists out of 9 criteria, a ✓, ○ or — in the row below a paper means, that the author(s) have either considered this topic, partially considered or did not consider it at all. The criteria are described in the following:

- **QoS Monitoring:** The authors consider QoS monitoring in their work, which means, they monitored VMs, the execution of services, ... in terms of their QoS attributes.
- **SLA:** If there is a checkmark (✓) for SLA, it means that the authors of the mentioned paper covered SLA-management in their work.
- **Elastic Processes:** If the authors mention EPs in their work, in terms of the three characteristics of elasticity defined in [17], a checkmark is set.
- **Cloud Computing:** If there is a checkmark in this line, it means, that the authors consider their Cloud computing as the research field in their work.
- **Reactive Optimization:** Reactive optimization means, that the presented system is optimized just in-time.
- **Proactive Optimization:** This means, that the resources, services, ... are optimized in advance.

- **Service Optimization:** Service optimization means, that the authors consider mainly services, i.e., the optimization aims to improve the service quality.
- **Business Processes:** This means, that the authors consider business processes or workflows in their work.
- **Reasoning:** Reasoning means, that the authors perform some kind of reasoning in order to optimize their system.

- **Automated Control in Cloud Computing: Challenges and Opportunities**
  Lim et al. address in [39] the topic of automatic controlling of computing resources in the Cloud. Their solution was based on a simple scenario in which a small startup company runs a Web application service that serves dynamic content to clients. The hosted application is horizontally scalable, i.e., it can grow to serve a higher request load by adding additional virtual machines. In order to do so, they followed an approach where a Cloud hosting provider runs its own control system to arbitrate resource requests, select guest VM placements, and operate its infrastructure to meets its own business objectives. For estimating the relationship between the CPU utilization and the cluster size under a synthetic heavy workload, a linear regression model was used. By applying this model they were able to calculate an estimated workload which is the minimum of CPU utilization but still be able to satisfy the clients demands. They claim, that using their formula it is possible to detect the right moment whether a new VM hast to be started or one can be shut down. While [39] focuses on the front-end tier of Web applications and simulated the actual controller implementation, Lim et al. continues this approach in [38] and focus on a the storage controller including the implementation of an elastic controller.
- **A Business Driven Cloud Optimization Architecture**
  In the paper [40], the authors Litoiu et al. discuss several facets of the optimization in Cloud computing. Further, they discuss the corresponding challenges and propose an architecture addressing them. This architecture supports self-management by automating most of the activities regarding optimization, like monitoring, analyzing, predicting, planning and executing. Within their architecture the authors consider different optimization goals of each layer because each layer reflects the layer's owner's economic interests, either to increase profit or

maximize end user satisfaction. While the focus of the presented work lays on the interests of different stakeholder, they are not really comparable to the approach followed within this thesis. However, the presented architecture reflects very well an interesting software architecture for Cloud computing.

- **Using Reinforcement Learning for Controlling an Elastic Web Application Platform**
  The authors Han Li and Srikumar Venugopal propose in [37] an approach for controlling an elastic Web application platform using reinforcement learning. They mention that the main challenge, "*manage the infrastructure so as to satisfy performance requirements of applications as well as minimize ongoing costs*" can be spit into two sub-challenges: the *provisioning* [66] problem (finding the smallest number of computing resources to satisfy the requirements) ands the *dynamic placement* problem [30] (distribute applications among the servers such that the applications are able to meet their response time and availability requirements). In their work, they used Reinforcement Learning (RL) [63] in order to control an elastic Web application hosting platform. Their platform is designed in two different kinds of VMs: first) a Proxy Server, which is responsible to forward the clients requests to the wanted service, and balance the load on each Backend Server. And two) a Backend Server, on which a application server hosting the service instances are deployed. In addition to that, the RL Controller is deployed on this VM.

  In order to "feed" their algorithm the virtual servers are monitored in terms of CPU usage. While this value directly influences the service quality which shows if a service is working as expected, they do not directly consider SLAs. While the focus of the RL algorithm lays on a local optimization, the result is a global optimized system. This means, each Backend Server is verifying if it is in a normal state. If it becomes critical, the RL Controller identifies an action for optimizing the system, like *Copy/Move* a Web service, or *Start/Terminate* a Backend VM.

  At this point we want to indicate that this work has influenced this thesis the most. While we are following a similar architectural design including similar actions for the optimization, we have different assumptions on the single Web Services. In the work of Han Li et al. each service instance is independent of each other, in our work, the focus lays on Business Processes, i.e., the reasoning component has to consider, that for processing a workflow, several Web services have to be invoked. Further, in ViePEP the optimization component uses global information, i.e., monitored data of each Backend VM in order to reason about an action, in contrast to the presented work, in which the RL Controller only uses locally monitored data.

- **A Novel Architecture for Realizing Grid Workflow using Tuple Spaces**
  Jia Yu and Raikumar Buyya addressing the topic in [80] of workflow and resource management within Grid computing. Since Grid computing can be seen as *the mother* of Cloud computing there are several similarities. So is the resource management within this grid as much important as it is in todays Clouds. In connection to the resources management challenge, the authors of [80] addresses the challenge of realizing a workflow management system in a Grid. To do so, they get use of the notification feature of a Tuple Space implementation. Using these notifications, they were able to realize a just in-time scheduling system, thus resources are only reserved when they are needed. All things considered, the authors of [80] are following a similar approach as we do in this thesis. However, a big difference is, that the system presented in the mentioned paper is not considering the future in the sense of how the system may look like, but it is reserving resources just in-time.

- **SLA-Aware Virtual Resource Management for Cloud Infrastructures**
  Hien Nguyen Van [68] proposes an automatic resource management system which aims at having the ability to automate the dynamic provisioning placement of VMs while taking into account, both application-level SLAs and resource exploitation costs. Further, this resource management system supports heterogeneous applications and workloads including both enterprise online applications with strict QoS requirements and batch-oriented CPU-intensive applications. The VM provisioning and packing problem was expressed using a two Constraint Satisfaction Problem.

- **SLA-aware Resource Management for Application Service Providers in the Cloud**
  Cardellini et al. [13] proposes an autonomic resource provisioning solution for Application Service Providers (ASPs). This solution enables ASPs offering a

Cloud-based application to handle the dynamic resources provisioning at application level by taking into account application specific QoS objectives as well as resource utilization costs. Their framework includes a Performance Monitor in order to detect SLA violations early and counteract while acquiring new computing resources. In order to do so, they on one hand act reactive, and on the other hand proactive, e.g., forecast the workload for the next time slot and whenever the resources are overloaded they acquire more. The difference to our solution is that, our focus lays on Business Process in combination of resources forecasting using this information. In contrast to that, Cardellini et al. focus on resource management by taking into account customers SLAs and QoS attributes.

- **Autonomic Virtual Resource Management for Service Hosting Platforms**
  Hien Nguyen Van et al. addresses in [67] the problem of autonomic virtual resource management for hosting service platforms with a two-level architecture which isolates the application specific functions from a generic decision-making layer. As Han Li et al. in [37] they split their optimization problem into two sub-problems: *provisioning* and *packaging* problem. However, in contrast to Han Li et al. and our implementation, the packaging problem addresses the placement of virtual machines on physical machines and not (web) services on virtual machines. They express this problem as a two Constraint Satisfaction Problem.

- **A min-max framework for CPU resource provisioning in virtualized servers using $\mathcal{H}_\infty$ Filters**
  Charalambous T. and Kalyvianaki E. suppose in [14] a solution for resource allocation in the Cloud. By using a $\mathcal{H}_\infty$ Filter Controller they try to minimize the maximum error in performance as measured by the requests mean response time. The difference to other approaches is, that they do not allocate new VMs but increase the resource amount of a overloaded one, i.e., if a the CPU load of a single VM is likely to go over a defined threshold, because of an increased workload, the $\mathcal{H}_\infty$ Controller allocates an additional CPU core to this VM.

- **A Profit-Aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers**
  The authors Wei Chen et. al [15] address the problem of VM deployment and reconfiguration from the view of Platform Providers. They identified the different

Table 3.1: Table of relate work (✓- covered, ○- partial-covered, —- not-covered)

	[13]	[14]	[15]	[37]	[39]	[40]	[67]	[68]	[80]	ViePEP
QoS Monitoring	✓	○	✓	✓	✓	○	✓	✓	○	✓
SLA	✓	—	○	○	—	○	○	✓	○	✓
Elastic Processes	○	—	—	—	○	—	—	—	—	✓
Cloud Computing	✓	✓	✓	✓	✓	✓	✓	✓	—	✓
Reactive Optimization	✓	○	—	✓	—	○	—	○	○	✓
Proactive Optimization	✓	✓	✓	✓	✓	○	✓	—	✓	✓
Service Optimization	○	○	—	✓	✓	—	—	—	—	✓
Business Processes	—	—	—	—	—	—	—	—	✓	✓
Reasoning	✓	—	✓	✓	✓	✓	✓	✓	✓	✓

concerns of Platform Providers in initial deployment and runtime configuration. The methods they proposed are mainly for reducing costs in both phases. In order to do so, they proposed a local adjustment strategy which reduces the number of VM migrations while balancing the multi-dimensional resource utilization according to the current requested resource demand or workload. Further, using a prediction model they used data from the runtime monitor in order to calculate the objective values and predict the future need of resources.

As far as we know, hardly anyone considers every criteria mentioned in Table 3.1. By looking at the presented related work and specially at the Table 3.1, we can see, that the field of Business Process was rarely handled, only [80] considers Business Process, but they do not consider them in connection with Cloud computing. Nevertheless, every presented work considers QoS monitoring in order to optimize their system either proactive, reactive or both. The lack of a solution, covering every presented criteria, led to the aim of this work: a Business Process Management System for elastic Business Processs in the Cloud, which is able to optimize the hosted services reactive or, by using a reasoning model, proactive.

CHAPTER 4

Requirements Analysis & Design

In this Section, we present the design of ViePEP. Before we do so, we describe an introductory example scenario which helps us to define the requirements for ViePEP. After the example scenario, we present the design of ViePEP which is split into four main parts: firstly, we derive the requirements from the example scenario, secondly, we describe ViePEP's life-cycle, and thirdly, we define the single parts of ViePEP that play an important role, such as the workflow management, the Backend VM on which the single services can be deployed, the communication part between each VM instance and, last but not least, we describe how the optimization is working.

In this Section we focus on the conceptual ideas of the framework, whereas details concerning the prototypical implementation can be found in Section 5.

4.1 Introductory Example Scenario

As a motivating example for ViePEP, we consider an imaginary bank which is an highly active stock market player (for sake of simplicity we call this imaginary bank IB).

IB provides several (software) services to their customers (to transfer money, to view account information, ...), to their employees (customer management, credit rating, stock market actions, ...) and some automated workflows such as the creation of back-ups. Many of these provided services consist out of several tasks, e.g., to create a credit rating for a customer several steps have to performed, such as checking the internal databases and requesting other banks or credit institutes for information about this customer. Another example is the automated back-up service which has to be run every night. This back-up service includes processes such as the collection of relevant data (log files, databases, ...), the preprocessing of the data and the creation of an archive or a replica at a different location. Eventually, every process provided by IB can be assigned to a single workflow consisting out of one or more different services (see Section 2.3).

These workflows are multifaceted, they consist out of different services and have different priorities. E.g., while some of the workflows have to be carried out in real time, others can be postponed into the near future. For determining whether a workflow has to be carried out before another, i.e., has a higher priority, it is necessary to prioritize them somehow, e.g., give each workflow a time limit until the it has to be finished.

IB is a big company and they want to be able to process hundreds of workflows simultaneously. Further they want to be able to scale their provided services in any dimension when needed. To be able to do this, IB moves their services into the Cloud. But still, they need a system which is able to fulfill the following requirements: they want to ensure that every workflow is processed in-time while considering the workflow's SLAs. Further, IB wants to minimize the amount of demanded computing resources in order to limit their expenses.

4.2 Requirements

The following requirements for the needed system arise from the introductory example scenario:

1. **Business Process Management System:** IB provides several BPs which have to be carried out automatically. That is why, the system they need has to be a BPMS. This BPMS has to fulfill the following requirements:
   1.1. **Workflow Managing:** The main part of an BPMS is a Workflow Manager, which is responsible to accept new workflows, schedule their executions, performing the executions while monitoring them.
      1.1.1. **Workflow Execution:** A component is needed which takes over the workflow executions. In order to know which service has to be executed within the workflow, a description is needed.
      1.1.2. **Workflow Description:** A workflow description represents a BP, i.e., it is now possible to process the workflow automatically. As mentioned in

[70] by van der Aalst et al., every workflow consists out of several steps which have to be carried out individually. Each single step is defined in the workflow description. Since it should be possible to prioritize the single workflows and services, additional SLAs can be defined.

1.1.3. **Scheduling:** It should be possible to schedule the queued workflows by sorting them according their priority. This can be done by using the SLA information provided within the workflow description.

1.1.4. **Workflow Queue:** In order to store waiting workflows, a data structure is needed where the workflows are queued until they get executed. This queue is ordered by the workflows' priority.

1.1.5. **Service Monitoring:** It is important to monitor the single service invocations in order to identify if everything is working as expected, e.g., ensure the given SLAs.

1.2. **Cloud Connection:** As mentioned above, IB wants to move their services into the Cloud. In order to get use of the features provided by the Cloud, the following is needed:

1.2.1. **Service Runtime Environment:** Each workflow consists out of several single services which have to be invoked. Therefore, a service runtime environment is needed in which the services can be deployed in order to be executable, e.g., an application server deployed on a VM.

1.2.2. **Resource Optimization:** Since IB wants to be able to scale-out to meet the required demand, in terms of acquired resources, it should be possible to acquire additional resources or releasing some during runtime, duplicate the services, and balance their executions on the available resources:

1.2.2.1. **Reasoning:** In order to find out when additional resources are required or when some can be released, because they are not needed anymore, a reasoning component is employed.

1.2.3. **Cloud-related Action Execution:** An interface to the Cloud is needed which is able to perform actions like creating additional VMs or terminating VMs.

1.2.4. **Service-related Action Execution:** In addition to the Cloud-related actions, it has to be possible to perform service-related actions to manage the services which are deployed in the service runtime environment. It has to possible to deploy a required service on a newly created VM. For that, a repository is needed, where every available service is stored.

1.2.5. **Service Repository:** The service repository is a database like structure where every service is stored. It has to be accessible from every service action executor.

1.2.6. **Load Balancing:** Since several instance of the same service can be available, a load balancer is needed which balances the invocations on the available resources.

4.3 System Life-Cycle

Now that we know what the requirements of the needed systems are, we have to define the system's life-cycle. Therefore we get use of the MAPE-K cycle as used in [31] for autonomic computing. The workflow can be seen in Figure 4.1. MAPE-K stands for *Monitor-Analyze-Plan-Execute* and *Knowledge* and its purpose is described as following:

- **Plan:** In the framework ViePEP two things have to be planned: 1) the execution of the workflows, and 2) the demanded computing resources (when to start how many VMs). For this purpose we have two individual components which are actually working together. The Workflow Manager, for scheduling the workflows' executions and a Reasoning Component, for computing the amount of needed computing resources.

- **Execute:** The Workflow Manager is not only creating a scheduling plan for the workflows' executions, but also performing the actual executions, i.e., invoking the individual workflow steps. Further, actions like starting and shutting down of VMs have to be executed.

- **Monitor:** The systems state is monitored in regular intervals (e.g., every 10 seconds) on two different levels : firstly, the load of acquired resources (e.g., CPU load, Ram load, network bandwidth) and secondly, the deployed service instances (e.g., service response time, failure rate, ...).

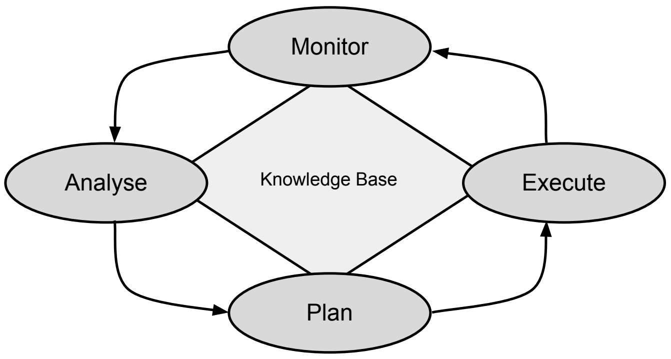


Figure 4.1: MAPE-K Cycle (adapted from [31])

- **Analyze:** The monitored data is analyzed in order to verify if SLAs have been violated and more resources are needed or if enough resources are available to process the workflows without any problems.
- **Knowledge Base:** The knowledge base is in our case a simple database located in a shared memory storing all the monitored data. In addition to that and for the purpose of a later evaluation, every executed action is also stored in this memory.

4.4 Component Description

4.4.1 Workflow Management

The key purpose of ViePEP is to process workflows. Therefore it is necessary to have a Workflow Manager. Its task is to manage the execution of the workflows. For this a queue-like data structure is holding the waiting workflows. Only the Workflow Manager is allowed to store new workflows in this queue, which means, that whenever a user requests to add a new workflow for being processed, the requested workflow will be stored in this queue. Since every workflow description can have optional SLAs, a detailed scheduling plan can be created. This scheduling plan is just an in-memory order of the workflows in the queue, thus the queue can be sorted accordingly. In addition, the Workflow Manager can react to the request of ad-hoc workflows. This means, that the Workflow Manager is able to accept new workflows at every moment during the system's runtime, update the scheduling plan and reorder the queue, thus workflows with a higher priority can be processed before workflows with a lower priority.

4.4.2 Service Deployment

In the case of ViePEP these service steps are represented by an arbitrary application. For now, we use Web services which can be invoked either via SOAP or REST and are deployed on a VM (we call this VM *Backend VM*). Since another purpose of ViePEP is to optimize the used resources and to ensure SLAs in order to provide a stable system, each single service and the VM it is running on, are monitored. To make it easier to dedicate the monitored data (e.g., CPU and Ram usage) to a single service instance (we define a running service as a *service instance*), we decided to deploy each single service on an own VM. Thus, we get one service per VM, but we can have several VMs running the same service. In any case, every VM is indirectly connected to each other using a shared memory in order to provide a place for storing the monitored data and to facilitate communication (see next Section). On account of letting ViePEP be autonomous, we decided that it should be able to start a new Backend VM whenever a new service has to be invoked. Therefore we created a service repository where every available service instance is stored.

4.4.3 Communication & Shared Memory

As mentioned in the previous Section, it is necessary to have a communication layer between the VMs. ViePEP should be distributed and very scalable, i.e., it should be possible to add and remove VMs during its runtime. Therefore, we need some kind of a shared memory which supports ad-hoc joining and leaving of peers. This shared memory has two purposes: first: serving as a communication layer, thus the VMs can communicate with each other, and second, as a database where the recorded data is read- and writable from each VM. In our case, we decided to make use of MozartSpaces[1] which is an Java-based open source implementation of the extensible virtual shared memory (XVSM) technology based on Tuple Space [34]. The communication is possible thanks to the notification feature provided by MozartSpaces. These notifications are based on a *Topic-based Publish-Subscribe* pattern: Clients (in our case the Backend VM and Action Executor) subscribe themselves on a topic and get notified whenever new information for that topic is available [21], e.g., every time a new monitoring entry from an Backend VM was stored or an action from the Action Executor was fired. As important as the communication layer is the possibility to store data in the MozartSpaces. The

[1]http://www.mozartspaces.org/

recorded data is saved in the MozartSpaces in a database manner. Records can be written and read in real-time which is very important for our optimization approaches. In addition to that, MozartSpaces provides the possibility to query the data similar to SQL. A more detailed usage of MozartSpaces is described in Section 5.4.

4.4.4 Optimization

As mentioned in the introductory example scenario, another key principle of ViePEP is the optimization of the computing resources, i.e., optimizing the load of the currently used VMs and the future needed ones. This means that we only want to have as much VMs running as we really need in order to guarantee every SLA and be cost-efficiently as possible.

At the first start of ViePEP, we just have one VM running: the BPMS, and whenever a new workflow request is coming in, the system checks if the needed services are already available. Afterwards, a scheduling plan is created and the executions are started (see above). In the case of having a specific service not already deployed on a VM, a new VM instance is acquired and the service is deployed on it.

As our basic idea is to provide a solution which is able to consequently adapt the system landscape, i.e., add or remove VM instances during the system's runtime since the Cloud user has to pay for every running VM. In our use case, hundreds or thousands of workflows are running concurrently and while they are executed, it is likely that more and more are started. As already mentioned, each workflow consists out of several service steps. These services are represented by applications that need to be invoked in order to execute the workflows. Executing these services leads to ever-changing demands regarding computing resources in terms of CPU usage and Ram usage.

There are several reasons why the needed resources within a given scenario or system landscape vary over time. It might be the case that very data-intensive tasks have to be carried out from time to time, services are invoked as regular batches, or simply that during a peak time (e.g., during the working hours) much more resources are needed than during of-peak time (e.g., at night).

With the possibilities of Cloud computing, in theory it is possible to have virtually unlimited resources. Thus resources can be added if the performance of a single service decreases or the resources can be released if the demand of requests decrease. In an ideal world with perfect information about the present and future situation, it would be easy to add as much computing resources as needed at a given point of time, leading to optimal cost efficiency. Unfortunately, we are not living in a world with perfect information.

Therefore, the challenge is, to use every available information about our system and the state it is in, for calculating the amount of needed computing resources. In fact, in the scenario described so far, two major sources are available: firstly, the current load of resources and secondly, the information about the current and future invocation of executed workflows (including the information about which particular services have to be invoked). We decided to use a *3 Step Optimization Approach* to solve this challenge:

1. **Step 1** In the first step the system tries to calculate the amount of needed resources for each single service. This is done, by learning from the monitored data. If our set of historical data is large and distinguishing enough, a data mining approach is able to derive for a single service invocation how the resource demand will look like.

2. **Step 2** As the next step, we get an overview about the complete future system landscape: Based on the information from the last step, and in combination about the information about the future workflows (and their single services), we can estimate the future need of resources in order to create a complete future system landscape. This is done for some predefined time intervals, e.g., 5 minutes.

3. **Step 3** When we get the information from Step 2, we follow an approach quite similar to the one proposed by Han Li et al. in [37], i.e., try to find the best action in terms of copy, merge, start, shutdown service instances and their VMs (or combinations of these actions). However, there is one major difference in the approaches and what we try to achieve: while the authors of [37] are doing a local optimization, we are optimizing on a system landscape level. While in a future version ViePEP will be able to optimize the workflow executions in the sense of being fast and effective, for now we concentrate on the optimization of the used computing resources.

For detailed information about this optimization please be referred to Section 5.2.3.

CHAPTER 5

Implementation

In this section our prototypical implementation of the ViePEP framework is presented. Firstly, we will have a look at the “big picture”, i.e. how the framework components are connected with each other. Subsequent to this, the details about their implementation will be explained step by step. The whole ViePEP framework is developed in SUN 1.6 Java [53] and can be built using Apache Maven 3 [22]. The system was developed and tested in a Linux environment. ViePEP was designed for the OpenStack [52], an open-source Cloud computing platform, but by the design of ViePEP it is possible to change or extend the implementation, thus ViePEP can be used within different Cloud providers, such as Amazon S3, Windows Azure, Google Cloud, ...

5.1 The Big Picture

As depicted in Figure 5.1 (using an FMC Block Diagram), ViePEP has five top level entities:

1. The **client** models service-based workflows and can optionally define SLAs. The workflows are modeled making use of an XML-based description format, as presented in Listing 5.1 which also defines non-functional constraints and preferences for each step, or for the whole workflow in form of SLAs. This description is handed over as a workflow request to the Workflow Manager (WfM) of the BPMS in order to instantiate and execute a workflow. A Client may request many

workflows at the same time and the system is able to serve several clients simultaneously.

2. The **BPMS VM** offers the central functionalities of a service infrastructure and Cloud control solution, e.g., workflow and service scheduling and load balancing. The BPMS is running inside a VM. We have decided so in order to allow for further self-adaptation capabilities in a future version of ViePEP and to ensure that in case of a high load of this VM it does not effect other services. The BPMS will be presented in more detail n Section 5.2.

3. A **Backend VM** hosts a particular service. In a typical ViePEP-based system, many Backend VMs exist at the same time and have to be controlled through the BPMS. Together with the BPMS, the Backend VM is the central entity in ViePEP and will therefore be presented in more detail in Section 5.3.

4. The **Shared Memory** is used to provide data sharing between the BPMS and the different Backend VMs. We chose MozartSpaces for this, as it allows to easily deploy and access a peer-to-peer-based, distributed shared memory. For a more detailed description refer to Section 5.4.

5. Last but not least, the **Service Repository** hosts service descriptions as well as their implementations as portable archive files, which allows to search for services and deploy them on a ViePEP Backend VM, see Section 5.5.

5.2 Business Process Management System VM

As already mentioned, the BPMS runs inside an own VM. For us it is very important to be able to easily scale our system. To have a centralized component for the workflow management part helps us to achieve this. A more detailed view of the components making the BPMS working can be found in Figure 5.2. The purpose of this VM is to plan and control the future executions of the workflows. The *Workflow Manager* is deployed for that reason, and in addition, it further delegates the actual service executions of these workflows to its helper component, the *Workflow Executor*. Further, a *Load Balancer* is deployed to balance the invocations and use the Cloud resources efficiently. Besides of the workflow management, a component for the resource optimization is deployed

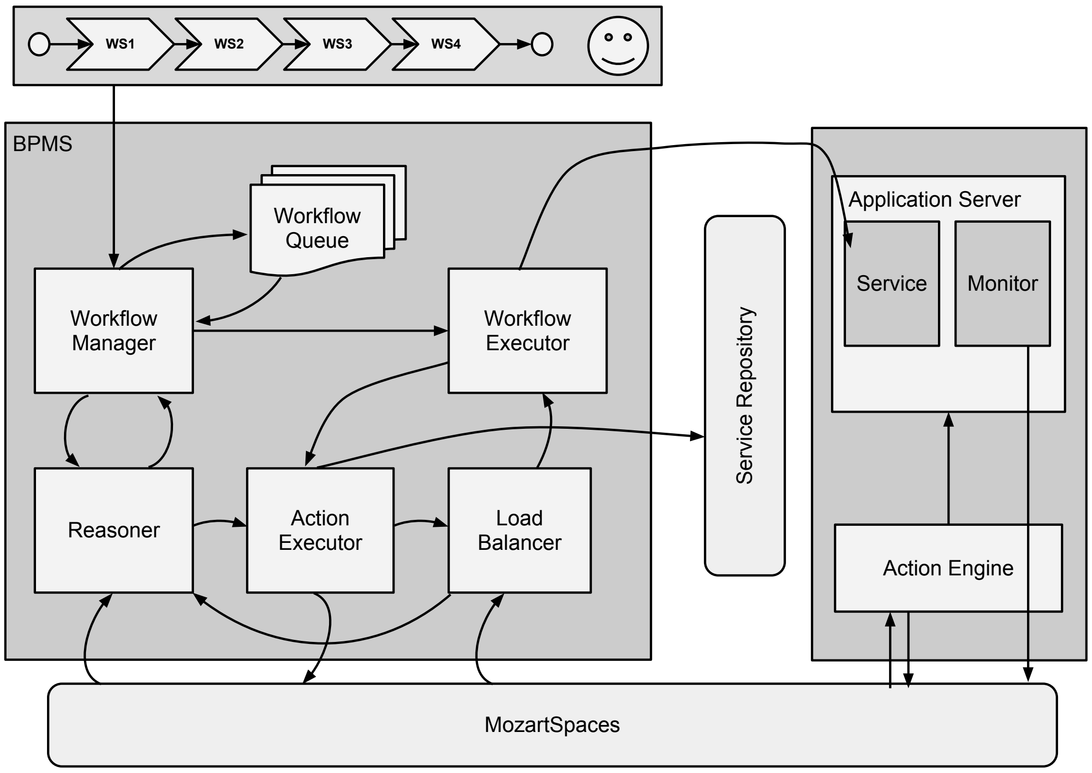


Figure 5.1: ViePEP Implementation

(*Reasoner*). For executing the actions coming from the *Reasoner* an *Action Executor* is deployed. Figure 5.3 shows a sequence diagram representing the workflow of the BPMS VM: A user requests ViePEP to start a new workflow. This workflow gets accepted by the Workflow Manager which re-schedule the waiting workflows and updates the queue. Further, the Workflow Manager processes the workflow queue by delegating the executions to the Workflow Executors. The Workflow Executor processes each workflow step by step, i.e., it *queries* the Load Balancer for the best fitting Backend VM and invokes the service. The result is stored for a later processing.

5.2.1 Workflow Manager

The WfM controls and schedules the workflows and further delegates the service executions to the Workflow Executors (see Figure 5.2) for their executions. It receives the necessary workflow description from the Client and stores it in the *Workflow Queue* for a

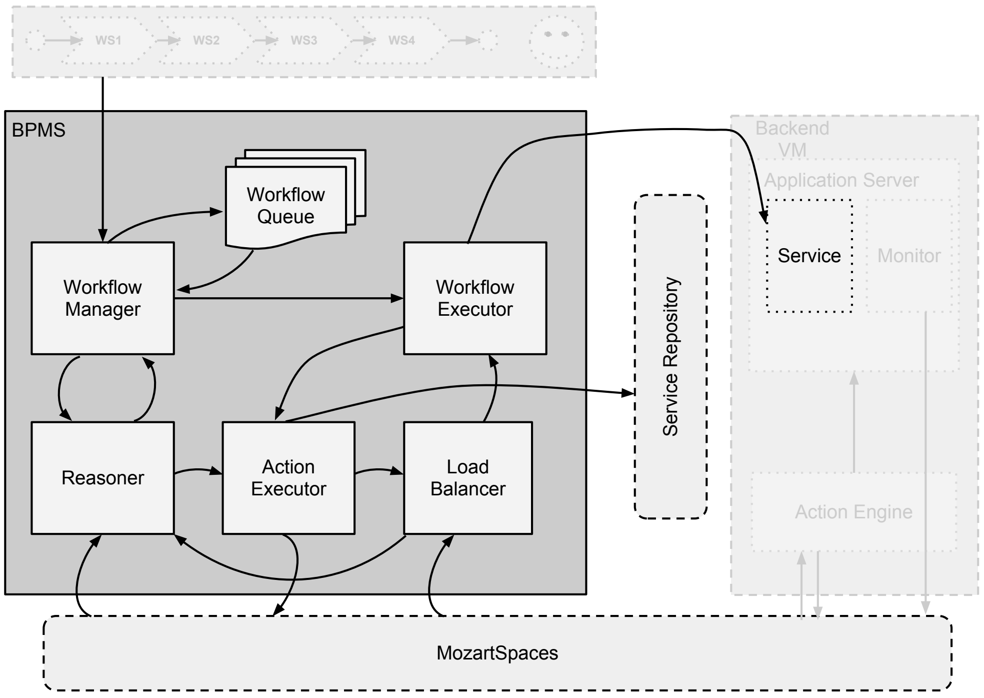


Figure 5.2: ViePEP: BPMS System Components

later (or immediate) execution. These workflow descriptions contain information about the single steps in a workflow and the accompanying SLAs. Afterwards, the Workflow Manager queries the Service Repository for the services matching the workflows steps. This mapping is needed in order to identify already running services instances through the Load Balancer. Based on this information, the WfM is able to issue service invocation requests to a particular Backend VM hosting this service and possessing enough resources to serve this invocation under given QoS constraints as defined by the SLAs.

There are many ways to describe workflows and SLAs [2, 42, 50]. However, in this work we make use of a basic lightweight workflow description language which can be found in Listing 5.1. The purpose of this format is to let users/ customers define their workflows and SLAs in one file which will be processed by our framework. Using this file, it is possible to describe a workflow (see element *businessProcess*) which has to have at least one process step (see element *stepsList*). So far, only sequential workflows are possible, but in a future work we will consider branching, loops, ... Both, the workflow itself and its process steps can have an optional SLA (see element *sla*) which can have the following SLOs:

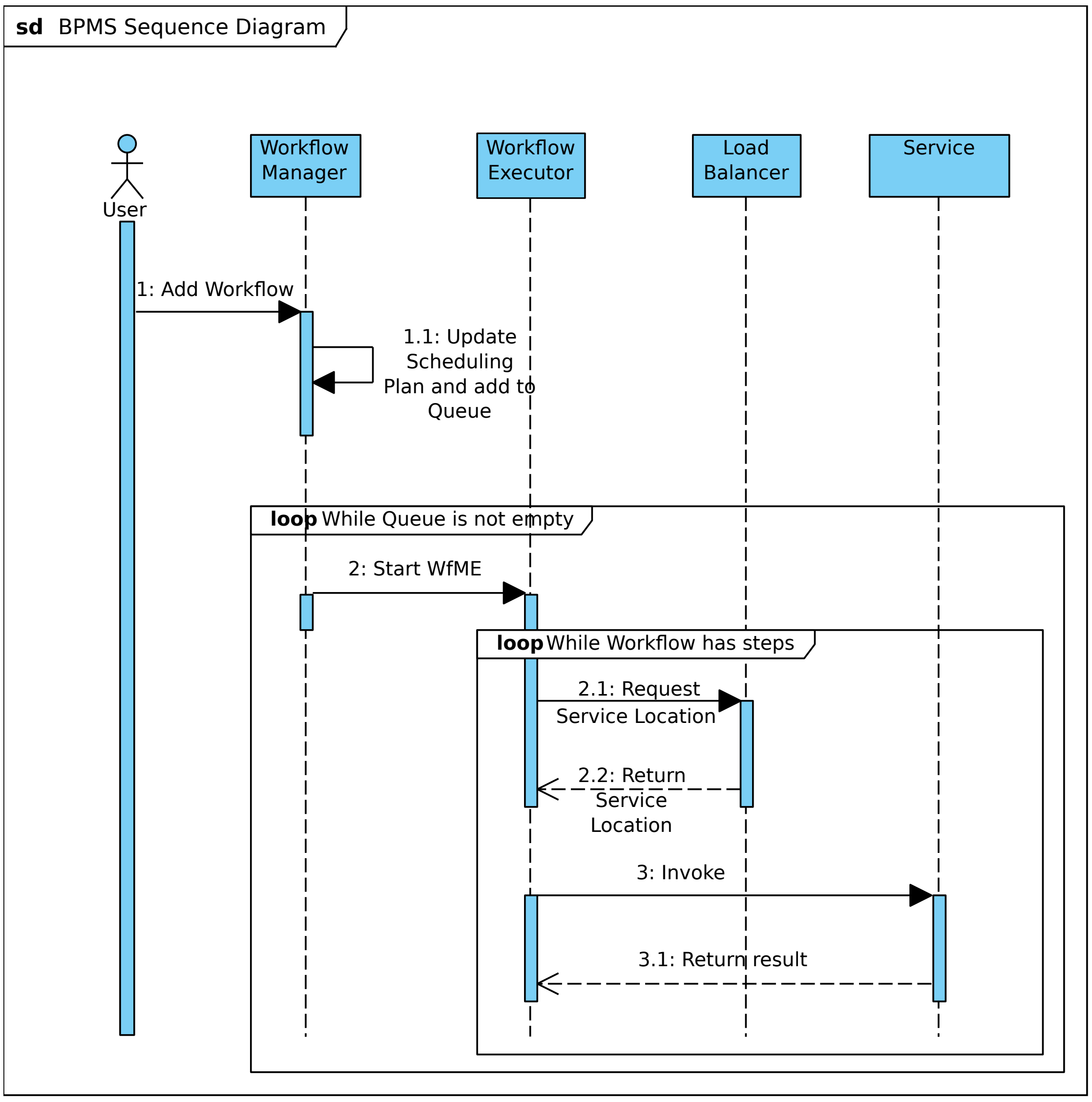

sd BPMS Sequence Diagram
User
Workflow Manager
Workflow Executor
Load Balancer
Service
1: Add Workflow
1.1: Update Scheduling Plan and add to Queue
loop While Queue is not empty
2: Start WfME
loop While Workflow has steps
2.1: Request Service Location
2.2: Return Service Location
3: Invoke
3.1: Return result

Figure 5.3: ViePEP: BPMS Sequence Diagram

- *endBefore*: This SLO describes the latest date and time the workflow's or the individual process step's execution has to be finished.
- *maxResponsetime*: The SLO maxResponsetime defines a service specific attribute. It says that the service response time should not be higher than the defined value.
- *maxCost*: Since every used Cloud resource produces costs, this SLO defines the

limit a user is willing to pay for the workflow or the process step.

A valid example can be found in Listing 5.2. In this example a workflow is defined which consists out of two steps: in step 1 a service with the service ID *webservice1* has to be invoked. Further, the user also defined a SLA which says that the service's response time has to be below 300 milliseconds. In step 2, a service with the id *webservice2* has to be invoked but the user does not care about when it is executed, but he defined a maximum cost for this service, which is 10€. In addition to the SLA on workflow step level, the user defined a SLA for the whole workflow: he wants this workflow to be executed not later than August 30th, 2013 at 09:00:00 am.

Anyway, by the nature of ViePEP, it is easy to replace our proprietary format with any non-proprietary format.

Listing 5.1: Business Process Description Language

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="↩
    qualified">

 <xs:element name="businessProcess" type="BusinessProcessType"/>

 <xs:complexType name="BusinessProcessType">
  <xs:choice minOccurs="1" maxOccurs="2">
   <xs:element minOccurs="1" maxOccurs="unbounded" name="stepsList" type="↩
       StepType"/>
   <xs:element name="sla" type="slaType" minOccurs="0" maxOccurs="1" />
  </xs:choice>
 </xs:complexType>

 <xs:complexType name="StepType">
  <xs:sequence>
   <xs:element name="step" minOccurs="1" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="serviceId" type="xs:string" minOccurs="1" maxOccurs↩
          ="1" />
      <xs:element name="description" type="xs:string" minOccurs="1" ↩
          maxOccurs="1" />
      <xs:element name="sla" type="slaType" minOccurs="0" maxOccurs="1" />
     </xs:sequence>
     <xs:attribute name="id" type="xs:NCName" use="required"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="slaType">
  <xs:choice minOccurs="0" maxOccurs="3">
   <xs:element name="endBefore" type="xs:dateTime"/>
   <xs:element name="maxResponsetime" type="xs:integer"/>
   <xs:element name="maxCost" type="xs:integer"/>
  </xs:choice>
 </xs:complexType>
</xs:schema>
```

Listing 5.2: Example Business Process

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<businessProcess>
 <sla>
  <endBefore>2013-08-30T09:00:00</endBefore>
 </sla>
 <stepsList>
  <step id="step1">
   <serviceId>webservice1</serviceId>
   <description>this is step 1</description>
   <sla>
    <maxResponsetime>300</maxResponsetime>
   </sla>
  </step>
  <step id="step2">
   <serviceId>webservice2</serviceId>
   <description>this is step 2</description>
   <sla>
    <maxCost>10</maxCost>
   </sla>
  </step>
 </stepsList>
</businessProcess>
```

To execute a workflow and its service steps, a *Workflow Executor* takes a workflow from the queue. Based on the workflow request and a workflow/service scheduling obtained from the Reasoner (see next point), the executor *queries* the Load Balancer for the best fitting service instances (Backend VMs) for the next workflow steps. For this we decided to use the Backend VM having a system load closest to a predefined threshold (For now, this threshold is a fixed value, but in a future version of ViePEP this will be dynamic and depends on the deployed service. For more information refer to Section 7.

The Workflow Executor is able to measure the service response time (this includes the network latency and the time it takes to generate a response from the time the service was invoked [57]). This is happening by simply measuring the time from the moment the execution was started to the moment the result is returned. This functionality is needed in order to determine deviations from the expected service behavior at an early state and avoid SLA violations, which can lead to penalty costs [36]. The information about the service invocations, i.e., date of invocation, the actual response time and the HTML result code is stored in the Shared Memory. If a deviation appears, a workflow re-planning is done by the WfM. If this replanning is done, the Reasoner takes the updated information about the current process landscape into account.

If the Workflow Executor does not receive a fitting service instance from the Load Balancer, i.e., no such service is available (this happens whenever a service has to be invoked for the first time), the Workflow Executor tells the Action Executor the missing service description and puts the actual workflow execution on hold, and puts it back to the queue. After that the Workflow Executor is idle for a few seconds and can start processing another workflow. However, ViePEP is designed in a way, that Workflow Executors never have to wait. Since a service instance can be unreachable because of an error, we had to consider this case. In any other case, the workflow will not be started before every needed service is deployed. The Action Executor queries in the meantime the *Service Repository* for the according service implementation and instantiates a new Backend VM. For that reason we have created Backend VM snapshots containing every necessary component. With help of this snapshot a new Backend VM can be started which is already fully configured. Depending on the Cloud provider this will take up to a few minutes. After that, the Action Executor deploys the appropriate service archive on this VM and tells an Workflow Executor that the according Web service is available, so that the paused workflow can be continued.

The Workflow Manager provides an interface allowing the Reasoner (or an system administrator in order to verify if ViePEP is running correctly) to get information about the number and the structure of the workflows (and their service steps) in the queue including their according QoS constraints and preferences. Furthermore, the reasoner can request via an interface the number of active Workflow Executors, active workflows, and services currently running on the system. Even more important, the WfM implements an interface allowing to request how many services have to be invoked at what point of time. In addition to that, information about new user-issued workflow requests as well

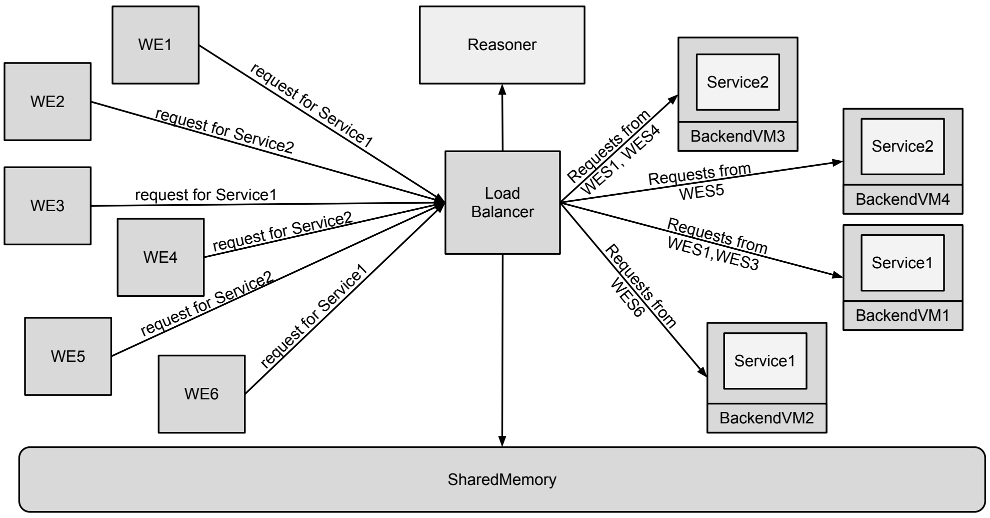


Figure 5.4: Load Balancer

as occurring and likely deviations in non-functional service behavior are provided to the Reasoner.

5.2.2 Load Balancer

As the name already implies, this component's purpose is to balance the load on the Backend VMs, thus it makes sure that the utilization of a Backend VM does not exceed a critical (upper) threshold. Importantly, the Load Balancer is a passive component. This means that when a service request comes in, it filters and provides information from the Shared Memory to the Workflow Executor and Reasoner, but it does not control the Backend VMs by itself. Figure 5.4 shows how the Load Balancer is working and with which other components it communicates: A Workflow Executor (represented in the figure by WS1-WS6) requests the wanted service from the Load Balancer. The Load Balancer retrieves this information from the SharedMemory (i.e., from the MozartSpaces) and delegates the service invocations to a service instance. By doing so, the Load Balancer takes care, that each service instance is used efficiently.

The Load Balancer provides an interface so that, it can get requested by the Workflow Executor in order to identify the best fitting Backend VM for a service invocation. For this, it retrieves the actual Backend VM states (in terms of occupied CPU and RAM resources) for the specified service from the Shared Memory and takes into account the scheduling information about current and future service invocations provided by the Reasoner. Based on this information, the Load Balancer is able to link the service requests from the Workflow Executor to a particular service instance running on a Backend VM.

In our prototypical implementation, the Load Balancer makes use of a rules-based approach to determine the *best* fitting Backend VM: Service requests are linked to the VM instance which degree of utilization is closest to a predefined upper threshold. Thus, the Load Balancer allows the Workflow Executor to invoke services until a VM's load reaches this threshold. If the Load Balancer is not able to allocate a service to a service instance, e.g., because all VMs are overloaded or there is currently no VM running this service, it gives this information back to the Workflow Executor, which then may conduct a re-planning as mentioned above or trigger a new reasoning.

The Load Balancer implements an interface, so that, it can be invoked by the Reasoner in order to determine the best fitting Backend VM for an action (see Section 5.2.3). For example, the Reasoner may decide that it is necessary to duplicate a Backend VM, as the service provided by the VM needs to be invoked too often as the current stock of VMs running this service would be able to handle it in a particular time span. In our case, the VM with the least load will be replicated, since the duplication action will produce some additional CPU load. As another example, the Reasoner may decide that a Backend VM is not required anymore as the requests to its hosted service can be handled by another Backend VM. In this case, the Load Balancer will select the VM with the least number of current service invocations; furthermore, it will effectively block further invocations of this service instance, i.e., if a Workflow Executor sends a particular service request to the Load Balancer, the VM to be terminated will not be regarded anymore. As it is a passive component, the actual command execution, to duplicate or terminate a VM will be issued by the Reasoner, not the Load Balancer itself.

5.2.3 Reasoner

While the BPMS controls the invocation of single workflows, i.e., it delegates the invocation to the Workflow Executor, the Reasoner aims to optimize the complete process and system landscape. Further, it is responsible to find a scheduling plan for the workflows and the included process steps under the given SLAs and cost, resource, and quality constraints. This plan is forwarded to the WfM, which itself will delegate this invocation to the Workflow Executor which invokes the services described in the single process steps. The scheduling is directly related to the controlling of the Backend VMs. For that reason we defined actions which are invoked by the Reasoner and performed by the *Action Engine* (see Section 5.3.2) (e.g., actions like the decision to start, terminate, and duplicate Backend VMs, move a Web service from one Backend VM to another, or exchange the services running on a particular VM). In any case, the Reasoner needs to take into account the information about the currently and future running workflows including their QoS constraints from the WfM. Further, It considers deviations from the expected workflow execution behavior in order to find an appropriate countermeasure. The Reasoner requests the information about the currently VM resources (CPU and RAM usage) from the Shared Memory and interacts with the Load Balancer in order to decide whether a particular Backend VM is sufficient to carry out a service invocation, or if another Backend VM hosting that service needs to be started (duplicated), or if a VM can be terminated due to low load.

5.2.3.1 Scheduling

The scheduling plan created by the Reasoner is based on the given SLAs for each workflow. We assume that each Backend VM has the same CPU and RAM configuration, and further, costs the same. Consequently every service invocation costs the same thus it does not matter where a single service instance is invoked. For now, the scheduling plan is only based on the SLO *endBefore*, thus workflows with an earlier target date have a higher priority. In the case, when two or more workflows have the same (or no) defined deadline, then they are handled by the *First-come First-served-policy*. The result is a *fixed* scheduling plan, which tells us exactly when how many services have to be invoked. This plan is needed for the resource prediction. While the SLO *endBefore* is needed for prioritizing the workflows, we always try to stick to the other SLO *maxResponsetime*. However, in a future version of ViePEP we will provide additional SLOs, e.g., penalty costs.

5.2.3.2 Prediction

As mentioned in Section 4.4.4, the reasoning is happening in three steps, first we calculate the need for one single service invocation, second, we try to forecast how the future need of resources will look like in the next few minutes in terms CPU and RAM usage, and third, we compute the amount of needed resources in VMs and perform the necessary action, e.g., starting/terminating VMs or moving/copying services. To calculate the need of resources for one single service invocation, each service is monitored (see sub-point of Section 5.3.1) in terms of CPU usage and RAM usage as mentioned before.

- **Step 1:** In the first step, we want to create a prediction model which is able to compute the need of resources for one single service invocation. But before we can do so, we have to filter and prepare the monitored data, since it may include some deviations and calculation errors. To create a prediction model which is not influenced by these errors we make use of *Oridinary Least Square (OLS) Linear Regression*. The reason why we use Linear Regression is that it attempts to find a straight line that *best fits* the data, where the variation of the data above and below the line is minimized while these deviations have a much higher influence to the average (see Figure 5.5). This graph shows the CPU load depending on the number of concurrent invocations. The square (including the line it sits on) was computed using OLS Linear Regression while the circle is the average value. As we can see, the average is more far away from the actual measured value than the predicted linear regression value.

  But before we can get use of the OLS Linear Regression algorithm, we need to sort the monitored data. In the current version, every VM looks the same, e.g., it has the same CPU and RAM settings (we neglect the deviation of these provided resources caused by the Cloud operating system). Therefore it is easy to bring the monitored data to a common denominator. In detail, the monitored data can be represented as a vector:

  $\{C_t, R_t, I_t, V, S\}$ where V is the ID of the VM on which the service is running specified by S. In addition to that, C_t defines the CPU load, R_t the actual RAM usage and I_t is the amount of concurrent invocations at time t (see Section 5.3.1).

  We sort this data by service IDs, thus we can remove the VM from the vector. This service specific monitoring data is filled into a graph, an example can be found in Figure 5.6.

  In the next step we try to find a relationship between these points by using OLS Linear Regression as described in [72]:

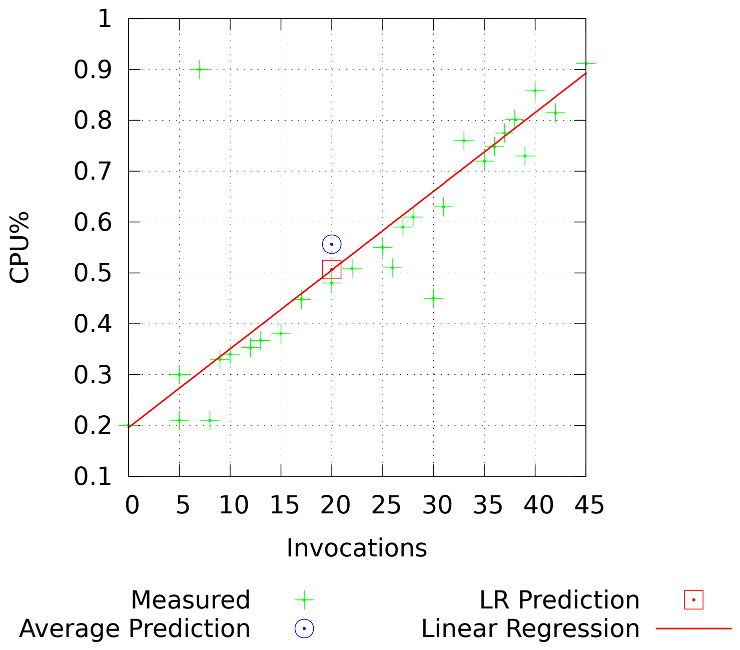


Figure 5.5: Linear Regression vs. Average Values

We have a sample with N observations on individual CPU loads and amount of invocations including other characteristics (e.g., RAM usage) from the monitored data of past invocations. We are interested in how the invocations are related to the CPU loads (and maybe other observation like RAM). Let us donate the CPU load by y and the other $k-1$ characteristics by $x_2, x_3, ..., x_K$. Our question is now: which linear combination of $x_2, ..., x_K$ and a constant can give us a good approximation for y (the CPU load). For that, we make use of an arbitrary linear combination which includes a constant:

$$\tilde{\beta}_1 + \tilde{\beta}_2 x_2 + ... + \tilde{\beta}_K x_K$$

$\tilde{\beta}_1 ... \tilde{\beta}_K$ are the constants to be chosen. If we index the observations by the variable i for $i = 1, ...N$ we can express the difference between an observed value y_i and its linear approximation as

$$y_i - [\tilde{\beta}_1 + \tilde{\beta}_2 x_{i2} + ... + \tilde{\beta}_K x_{iK}].$$

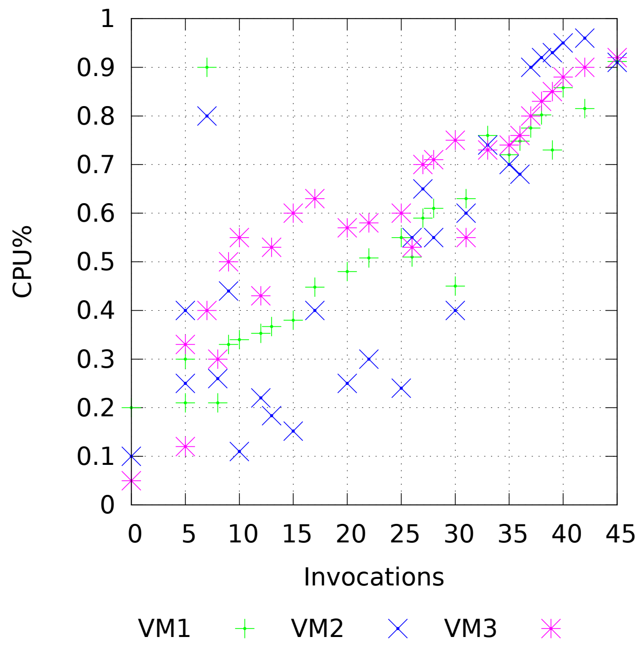


Figure 5.6: Combined Graph for 3 VMs

In order to simplify this derivations we collect x-values for an individual i in a vector x_i including the constants and introduce the short-hand notation.

$$x_i = (x_{i2}\, x_{i3}\, ...\, x_{iK}).$$

After collecting the $\tilde{\beta}$ coefficients in a new K-dimensional vector $\tilde{\beta} = (\tilde{\beta}_1, ..., \tilde{\beta}_K)'$ we can write :

$$y_i - [\tilde{\beta}_1 + \tilde{\beta}_2 x_{i2} + ... + \tilde{\beta}_K x_{iK}] \text{ as } y_i - x_i'\tilde{\beta}.$$

It is desirable to choose $\tilde{\beta}_1 ... \tilde{\beta}_K$ such that, the differences are as small as possible. The most common approach is to choose $\tilde{\beta}$ such that the *sum of square* is as small as possible. Therefore, we determine $\tilde{\beta}$ so that it minimizes the following function:

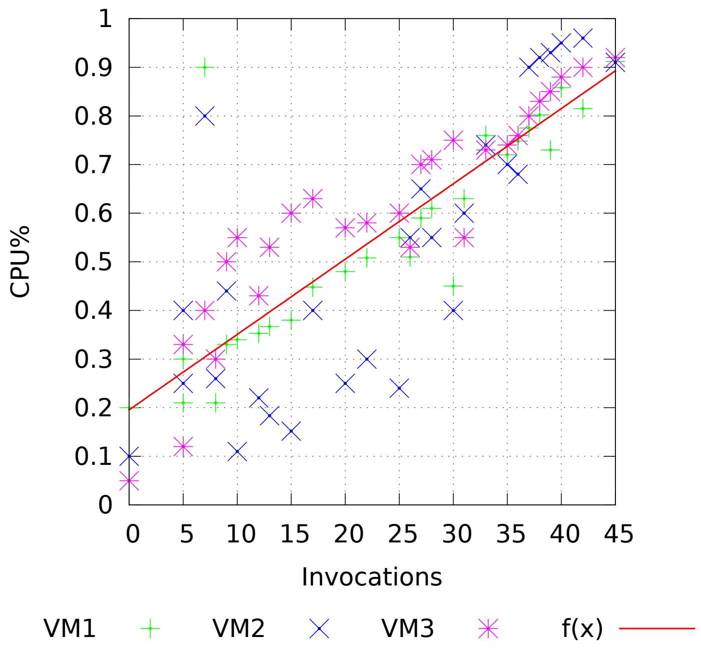


Figure 5.7: Combined Graph for 3 VMs (including Linear Regressions Line)

$$S(\tilde{\beta}) = \sum_{i=1}^{N} (y_i - x_i' \tilde{\beta})^2$$

Taking the square ensures that positive and negative deviations do not cancel out when taking the summation. In our current version we use Apache Commons Math's implementation to solve the OLS Linear Regressions [3]. The result of the linear regression analysis for Figure 5.6 can be found as a line in Figure 5.7.

- **Step 2:**

  The result of Step 1 is a function in which we can fill the amount of invocations in the near future. The result is the amount of needed CPU resources which will be forwarded to Step 3.

  In detail: The Reasoner requests the Workflow Manager for the queue of waiting workflows including a list of currently running workflows. These workflows are split into its single service steps thus it can add up the amount of invocations for each single service. The result is a plan of *which service* has to be invoked *how often* in the next minutes. This can be again represented as a vector: $\{S, I\}$ where S is the service ID and I the number of future invocations. The amount of future invocations is filled into corresponding Regression Function from Step 1 and the approximate amount of needed CPU usage can be computed. Since we assume that each VM can accept as much requests until a upper threshold is reached (at the moment we have a fixed threshold of 80%), we can compute how many Backend VMs we need at least for each service instance.

- **Step 3:**

  In Step 3 the result of Step 2 (the amount of needed Backend VM for each service instance) is compared to the actual systems state and a necessary action (see Section 5.3.2) is performed.

  In detail: The Reasoner queries the Shared Memory in order to retrieve the information about how many Backend VMs are up and running and *asks* the Load Balancer if any Backend VM is currently locked, e.g., it is currently performing an action. Out of this information it calculates the available CPU resources which are available for each service instance. By comparing this to the result of Step 2, the Reasoner knows whether it is necessary to duplicate a Backend VM or if one can be terminated. This decision (action) is forwarded to the Action Engine which handles the actual execution.

5.3 Backend VM

While the BPMS VM controls the process and system landscape, the actual service execution is done on the *Backend VMs*. Each Backend VM provides Software-as-a-Service (SaaS) in terms of a particular Web service. As can be seen in Figure 5.8, a Backend VM features two major components: The Application Server (*Service Deployment)* and the *Action Engine*.

5.3.1 Service Deployment

In order to host Web services, a Backend VM needs an Application Server capable to run it. At the moment, we employ Apache Tomcat[1]. However, it is possible to switch

[1]http://tomcat.apache.org/

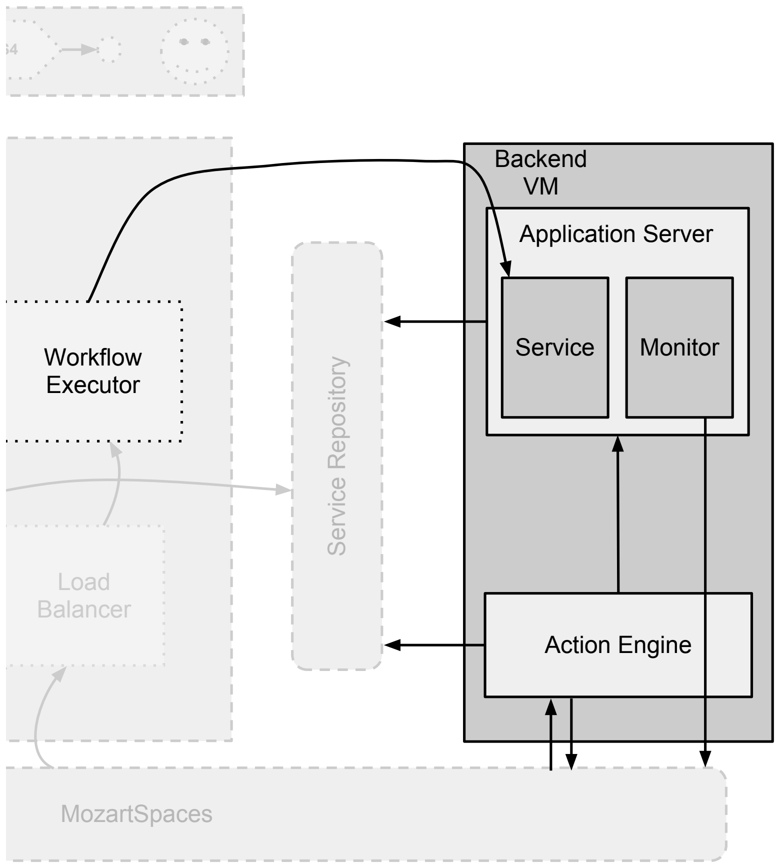


Figure 5.8: ViePEP – Backend VM components

to any other J2EE application server like Glassfisch[2] or JBoss[3]. The Application Server comprises two components, namely the actual *Service Instance* and a *Service Monitor*:

- **Service Instance:** As mentioned above, services are generally stored in the Service Repository. To host a service within the Application Server, the Action Engine retrieves the according Web Application ARchive (WAR)-file from the repository and deploys it on the Application Server. In the current version we support any RESTful Web service which can be called using an HTTP GET request or can be invoked using a remote procedure call.

[2]http://glassfish.java.net/
[3]http://www.jboss.org/

- **Service Monitor:** As explained above, the BPMS makes use of information about the Backend VM's resources in terms of CPU and RAM utilization. Hence, ViePEP-enabled Backend VMs feature an Application Server Monitor. Monitoring is conducted on a Platform as a Service (PaaS) level, i.e., the CPU and RAM utilization is measured for the VM, but not the underlying hardware/infrastructure. We use psi-probe[4] as server monitoring tool. It generates an HTML report for the service running on the application server which is parsed by our monitoring component. This report contains information about the number of invocations happened in total and the service response time. To get the information about the Backend VMs CPU load and RAM usage, a simple bash script is executed. The recorded data can be represented in an vector:

  $$\{C_t, R_t, I_t, V, S\}$$

  V is the ID of the VM
  S_V specifies the service instance running on V
  C_t defines the actual CPU usage in percent at time t
  R_t defines the actual RAM utilization in percent at time t
  I_t defines the number of concurrent invocations at the time t

  The monitored data is stored in the Shared Memory in order to be available for every other component including the Reasoner.

5.3.2 Action Engine

The Action Engine is responsible to perform commands to the Application Server and to its hosting Backend VM. It receives the according commands from the Reasoner through the Shared Memory. The most important commands in the context of the work at hand are [37]:

- **START** a new Backend VM: An empty VM is started by the Reasoner by issuing a corresponding command to the Cloud infrastructure hosting the Backend VMs (in our case: OpenStack; not depicted in Figure 5.1). When the Backend VM has successfully booted, the Action Engine obtains the needed service file from the Service Repository and deploys it on the Application Server.

[4]http://code.google.com/p/psi-probe/

- **TERMINATE** the Backend VM. Again, the according action command was issued by the Reasoner. If the Action Engine receives the command to terminate itself, it first requests information about currently running service invocations from the Application Server. If there are any, the Action Engine waits and regularly polls the Application Server until all service invocations have been finished. After that, the Action Engine unregisters the Backend VM by pushing the according status information to the Shared Memory, and finally terminates the VM. Afterwards, the Reasoner and Load Balancer will not take this Backend VM into account anymore (in terms of scheduling and service invocations).

- **DUPLICATE** an existing Backend VM. If the Reasoner determines that the computing resources hosting a particular service are not sufficient to carry out the future service invocations, it can issue an action command to a Backend VM in order to duplicate it. For this, the Action Engine will start a new Backend VM which hosts the same service.

- **EXCHANGE** the deployed service by another service. In some cases, the hosted Web service is not needed anymore, as there will be no further invocations in the (near) future for that service. However, another service needs to be started. Since the starting of a new VM takes some time, we can speed up the deployment of service by reusing the former Backend VM by exchanging the service running on it. To exchange a service, the Action Engine behaves similarly to the termination of a Backend VM: First, it checks if the provided service is currently invoked; if this is the case, the Backend VM waits until the invocations are finished and blocks every further invocations. Second, the current service is replaced by another service from the Service Repository.

- **MOVE** a running service to another Backend VM: The Action Engine is able to move the whole system state to another server. Again, running service invocations need to be finished first and no further invocation is accepted before this action command can be performed.

So, whenever the Reasoner decides, that an action has to be performed, it first communicates with the Load Balancer in order to find the *best* fitting Backend VM. The *best* fitting Backend VM is always related to the action command which has to be performed, for this purpose refer to Section 5.2.2.

5.4 Shared Memory

The *Shared Memory* is used to provide data sharing between the BPMS and the different Backend VMs. We chose MozartSpaces[5] for this, as it allows to easily deploy and access a peer-to-peer-based, distributed database [33]. It further enables ad-hoc joining and leaving from peers (in our case Backend VMs). Mozart Spaces provides the possibility to query for data in the database. This is very useful for the Reasoner, thus the data returned by the query is already sorted. In addition to the usage as a shared memory, an important feature coming with Mozart Spaces is the possibility to raise notifications: We need this communication method in order to provide a reliable communication layer, thus we can send commands from the *Reasoner* to the *Action Engine*. As it is presented in Figure 5.9 the shared memory can be accessed from the *Reasoner*, the *Load Balancer*, the *Action Executor* and the *Action Engine* on the Backend VM.

- The **Reasoner** has read-write permissions: it can read from the Shared Memory, i.e., to retrieve some more information about the services, and is able to write something to it.
- The **Action Executor** has a write-only permission. Whenever the Reasoner *decides* that an action has to be performed, it delegates this action to the Action Executor. This one writes the corresponding action into the shared memory, thus a notification is raised at the target Backend VM.
- The **Load Balancer** has a read-only permission: it gets notified whenever new monitor data is available, thus it has the actual information about the whole system landscape, so that it can tell the Action Executor and/or the Workflow Executor about the best Backend VM for their needs.
- The **Action Engine** has read and write permissions: it registers a notification listener on the shared memory with the service ID which is deployed on its host VM. Thus, it gets informed whenever the Action Executor sends an action command. In addition to that, it is able to write into the space, e.g., right after the Backend VM has been started, it registers itself in the shared memory, and after it received a shutdown message it unregister itself from the shared memory.

[5] http://mozartspaces.org

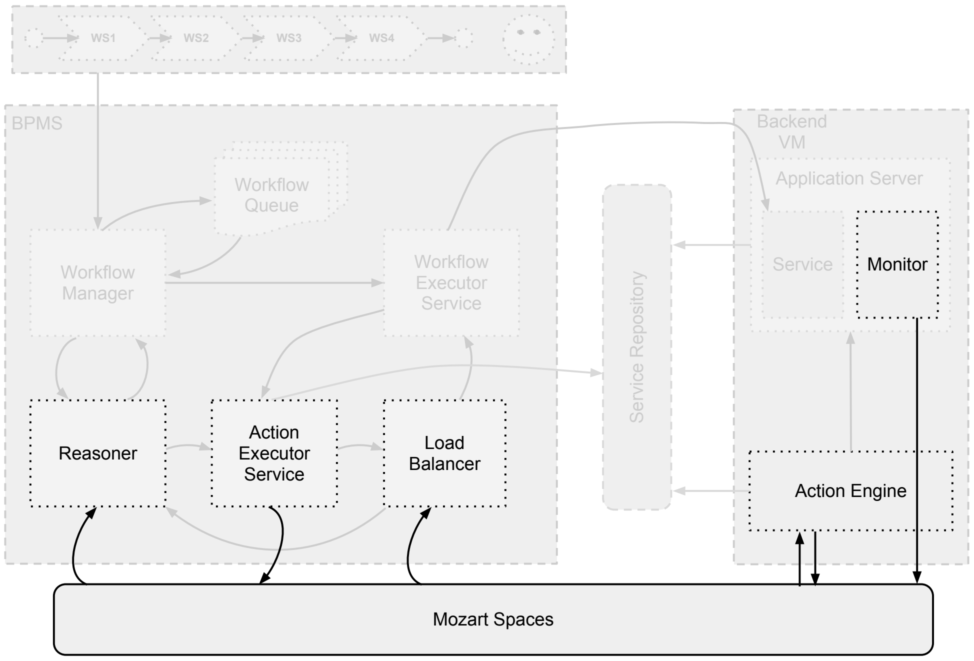


Figure 5.9: ViePEP – Shared Memory

5.5 Service Repository

The *Service Repository* hosts the service descriptions of the Web services as well as their implementations as portable archive files (i.e., as WAR-file). It provides a search mechanism, thus the Action Engine/Action Executor can query this repository in order to find a service they want to deploy on an arbitrary ViePEP Backend VM. Figure 5.10 shows which components can interact with the Service Repository.

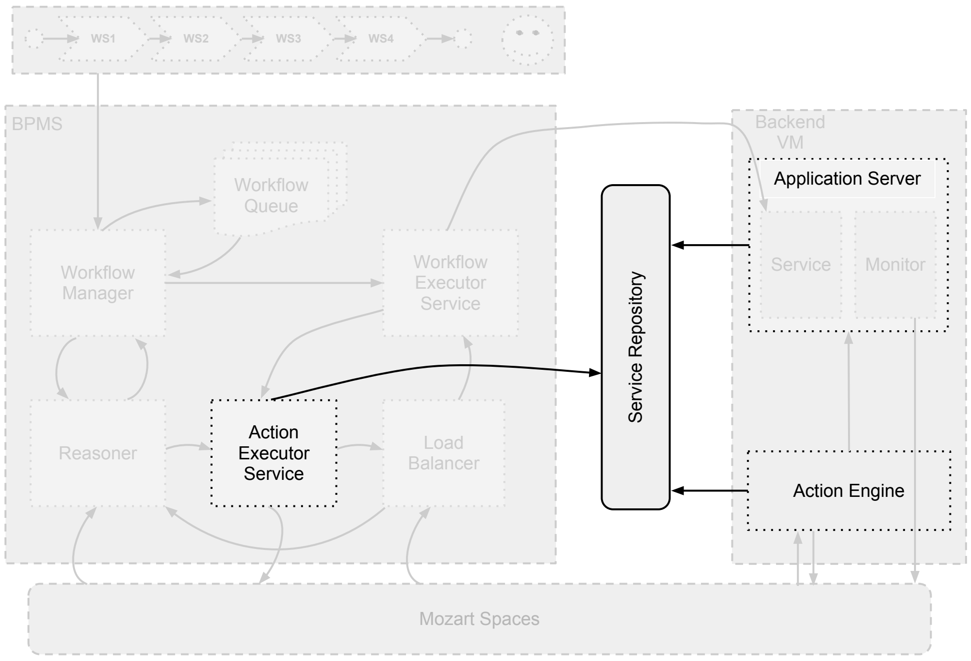


Figure 5.10: ViePEP – Service Repository

CHAPTER 6

Evaluation

In this section we perform an evaluation of the work presented in this thesis. The evaluation focuses on the design decisions discussed in Section 4 as well as the concrete prototype implementation presented in Section 5. While the ViePEP framework can hardly be compared to existing similar solutions on the whole, we compare our prototype including the optimization against a baseline. In our case the baseline uses the same setup but without an optimization, i.e., the system works reactive: whenever the system is not able to handle the requests for a single service, a new instance will be created.

6.1 Evaluation Scenario

6.1.1 Scenario

For evaluating ViePEP, we decided to use the following scenario setup from the same domain as our example from Section 4. For the sake of simplicity and to let the results be classifiable to the performed actions, we decided to make use of one single business process. The business process we chose consists out of five different steps and can be found in Figure 6.1. This business process consists out of 5 steps, first: a dataloader service, second, a pre-processing service, third, a processing service, fourth, a reporting service and fifth, a mailing service. To simplify matters, we simulate the operations which are usually performed in such services, however, the actions usually performed

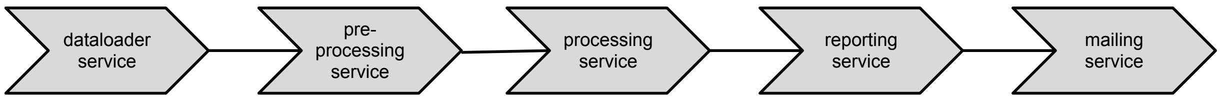


Figure 6.1: Evaluation Workflow

in such a service a described below. In order to get reliable results we decided to run the presented business process 20,000 times:

1. **Data Loader Service:** The *Data Loader Service* simulates the loading of the needed data from an arbitrary source. This could be a service which reads data from a database, from sensors or a service which communicates with other services. However, since this service is not performing any calculations, this service needs not many resources.

2. **Data Pre-processing Service:** After the data has been loaded, it needs to be pre-processed before it can be used in another service. This pre-processing is a bit more resource-intensive than loading the data. In order to simulate this load, we perform some simple mathematical calculations every time this service is invoked.

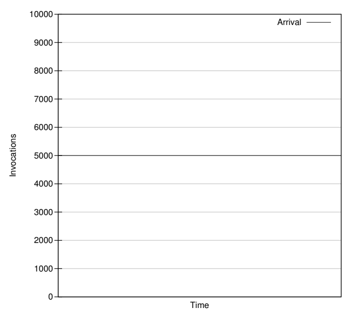
3. **Data Processing Service:** After the data has been prepared for processing, this service is called. It simulates the processing of the data. This service needs in contrast to the other services more computational resources, especially the CPU usage will be much higher than for the others. In order to simulate a high CPU usage we perform some complex resource intensive mathematical calculations, each time the service is invoked, i.e., calculating the Fibonacci sequence for a few seconds.

4. **Report Service:** After the data has been processed a report will be generated, e.g., in form of a PDF file. The needed CPU usage for generating such a report depends highly on the data being processed in the step before. In order to simulate this CPU load, we perform again some simple mathematical calculations as done in the *Data Pre-Processing Service*.

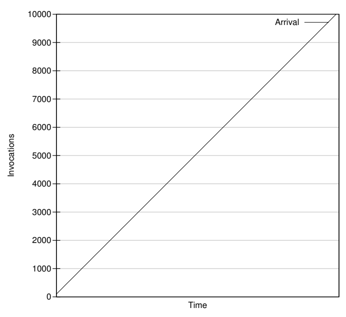
5. **Report Mail Service:** At the end of our example business process, the created report has to be delivered. This can be the sending of an email or pushing the created report on a web server. However, this service is again not resource intensive,

therefore, we make use of the similar mathematical calculations as in the *Data Loader Service*.

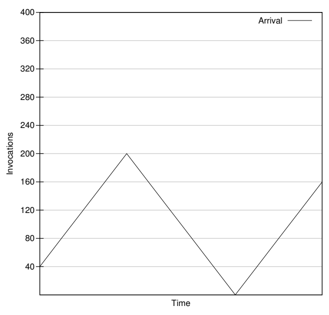
6.1.2 Arrival Functions



(a) Burst Arrival



(b) Linear Arrival



(c) Pyramid Arrival

Figure 6.2: Arrival Functions

For our evaluation we decided do use three different arrival functions as can be seen in Figures 6.2a, 6.2b and 6.2c. In condition to that, each arrival function will be run three times thus we get reliable data:

1. **Burst Arrival Scenario:** In this scenario we execute the workflows in a burst manner, i.e., the same amount of simultaneously executed workflows will be started every few seconds. For the time steps we decided to use an interval of 10 seconds. The graphical representation of this function can be found in Figure 6.2a. The number of simultaneously executed workflows is set to 150. However, we assume that this value is not relevant to the result of our optimization which will be clarified in the discussion below.

2. **Linear Arrival Scenario:** In this scenario, the workflows are executed in the manner of a linear rising function, i.e., $y = k * x + d$ where y is the amount of simultaneously executed workflows. Since a continuously increasing execution of the workflows would be limited by the underlaying operating system, we decided to increase the number of simultaneous workflow invocations by 10 every 5 seconds. The Linear arrival function can be found in Figure 6.2b.

3. **Pyramid Arrival Scenario:** In this scenario we execute the workflows in form of a Pyramid Function which can be seen in Figure 6.2c. In this scenario we start with a low number of simultaneously executed workflows increase it to a maximum and decrease it again to 0. We repeat this steps until the workflow queue is empty. This scenario is used to test the behavior of the optimization in case of seemingly unpredictable arrival pattern.

6.1.3 Baseline & Scenario

As shortly mentioned above, for evaluating our scenario we define the baseline as following: We make use of ViePEP but disable the optimization. This means, that no reasoning is performed in order to acquire additional resources. Since it would be quite unfair to have just a limited amount of resources available, we decided to introduce an additional functionality, thus additional resources for a particular service instance are acquired whenever the load of this instance exceeds the threshold. However, since there is no further optimization, the load gets balanced at all Backend VMs evenly. That is why, a Backend VM will never be released again.

We assume that in each scenario, our approach will perform better than the baseline, because of the following three facts: First, since we make use of a prediction model in our approach, we are able to forecast the amount of needed resources and acquire them in advance. In contrast, the approach without optimization will only acquire additional resources, whenever the load of one single service instance gets to high, this may lead to just shifting the bottlenecks, i.e., when starting the experiments, the first service will only be able to handle an limited amount of service invocations before the VM gets overloaded. As soon as the VM is overloaded, it will be duplicated, thus it can handle more invocations. Since the first service is able to handle more requests now, the second service has to handle more requests as well, consequently, it gets overloaded as well and needs to be duplicated. The same applies for the third, fourth and fifth service. Second, although we will try to forecast the amount of future resources, it may happen, that we have acquired too many resources. As soon as ViePEP realizes that too many resources are available, i.e., some of them are not needed at all, ViePEP will start to release them, thus only exactly as much resources are acquired as needed. In contrast to that, in the baseline configuration, no optimization is happing, that is why the unneded resources will not be freed again. Third, because of our prediction model, the needed resources are acquired in advance, thus no bottlenecks will arise. This makes it possible for ViePEP to be much faster than the approach without optimization.

Our evaluation workflow scenario can be found in Figure 6.1 and is described in Section 6.1.1. Therefore, we have 20,000 workflows, each consisting out of 5 service steps. In order to see how our system behaves under different premises, we make use of 3 different arrival patterns described in Section 6.1.2 and presented in Figure 6.2. During the development, we noticed, that starting a Backend VM the first time will take more time, than duplicating it later on. This can be traced back to the fact, that when instantiating an image the first time, the underlaying Cloud operating system needs to copy the Backend VM's image to a new physical machine. This takes longer than duplicating this image on the same physical machine. Because of this, we decided to start our experiments with one VM for each service instance available. As soon as every instance is available, we start our evaluations according the arrival patterns and measure their executions according the measurements metrics described in the next Section.

6.2 Measurement Metrics

After we executed each scenario 6 times (3 times with optimization and 3 times without optimization) we examine the results under the following measurement metrics and compare it to each other.

- **Duration:** The first criteria is the overall execution duration of all 20,000 invocations, i.e., the timespan from the start of the first workflow execution to the moment when the last workflow execution is done.
- **Total Needed VMs:** The second criteria is the total amount of needed VMs including their lifetime, i.e., the result of this is a list of VMs and how long they have been running.
- **Costs:** The resulting costs are calculated by using the following approach: Since we assume that each VM costs the same, we can add up the overall runtime of the VMs. In this criteria we do not consider any extra costs which may rise when a new VM has to be started, i.e., the costs for 20 VMs running for 5 minutes are the same as for 10 VMs running for 10 minutes. Further, we do not consider the different costs for different services since this would be needlessly complicate

Table 6.1: Evaluation Results (average of all 3 runs)

	Burst Arrival		Linear Arrival		Pyramid Arrival	
	NoOpt	Opt	NoOpt	Opt	NoOpt	Opt
Queued Business Processes	20,000	20,000	20,000	20,000	20,000	20,000
Duration in minutes (std. deviation)	93.33 (σ=2.89)	76.67 (σ=2.89)	71.67 (σ=5.77)	60 (σ=2.89)	66.67 (σ=2.89)	65 (σ=0)
Highest Concurrent VMs (std. deviation)	8.33 (σ=0.57)	9.33 (σ=0.57)	16 (σ=0)	16.33 (σ=0.57)	15.67 (σ=0.57)	12.67 (σ=0.57)
Costs in VM-Minutes (std. deviation)	745 (σ=66.14)	526.67 (σ=11.55)	915 (σ=104.04)	758.33 (σ=53.93)	831.67 (σ=28.43)	576.67 (σ=15.28)
Lost Invocations (std. deviation)	116.6 (σ=13.50)	31 (σ=1)	120.3 (σ=3.51)	27 (σ=3.61)	97.33 (σ=2.52)	45.66 (σ=3.79)

the evaluation. To give the costs an independent value we define the unit *VM-Minutes*. 1 *VM-Minute* is defined by 1 VM running for 1 minutes, therefore the overall costs for 20 VMs running for 5 minutes is *100 VM-Minutes*.

- **Lost Invocations:** The next criteria is the amount of lost invocations. Although our system is designed in a way that a failed invocation has to be repeated it may happen that a service invocation may be interrupted because of an unforeseen event. The result of this is the failed invocations rate.

6.3 Results and Discussion

In the following we present the results of our evaluations. Since every scenario was executed 3 times, the following values are the average values of the 3 executions, i.e., $\frac{1}{3} * \sum_{i=1}^{i=3} x_i$ where x_i is the recorded value.

The detailed results can be found in the Appendix: for the burst arrival scenario results see Figure A.1, for the linear arrival scenario results see Figure A.2 and for the pyramid arrival scenario see Figure A.3.

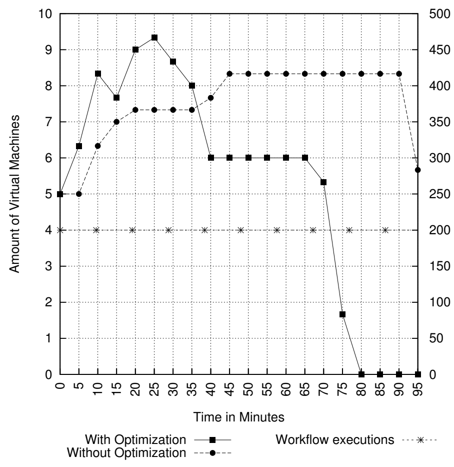


Figure 6.3: Burst Arrival Results (average of all 3 runs)

6.3.1 Burst Arrival Scenario

Figure 6.3 shows the results of the burst arrival scenario experiments, i.e., it shows the average of all 3 runs with optimization and the average of all 3 runs without optimization.

1. **Without Optimization:**

   We had a maximum of 9 active VMs running simultaneously and an average runtime of 95 minutes. In total we had 5 VMs running for ~100 minutes, 1 VM running for 90 minutes, 1 VM running for 80 minutes, and 1 VM running for 60 minutes. This leads to the overall total costs of 730 VM-Minutes.

2. **With Optimization:** Figure 6.3 shows also the results of the burst arrival scenario with optimization. At the beginning we started with 1 VM per service but already after a short timespan of 5 minutes, the first two services, i.e., the dataloader and preprocessor service have been duplicated, thus it can handle a higher load. Because of that we achieved a higher throughput which led to the need of a duplication of the other services. After about 40 minutes, the system found the perfect amount of needed Backend VMs. It held this state for the rest of the experiment, but as soon as the resources are not needed anymore, they are freed.

   The experiment's results can be found in Table 6.1. The costs for the first run are calculated like the following (the same takes effect for the other two runs): We had

5 VMs running for the whole 70 minutes. In addition to that, the system started an additional VM for the dataloader and pre-processing service after 5 minutes for 20 minutes. Further, the system started a second VM for the report service for about 20 minutes and a second VM for the mailservice for 10 minutes. Again, it is obvious that the process service is the one with the highest need of resources, therefore, the system started an additional VM for 65 minutes. If we sum this up, we get the the total costs of 465 VM-Minutes.

All in all, the results of the burst arrival scenario are for all iteration pretty much the same. The evaluations *With Optimization* and *Without Optimization* are for the first few minutes of the experiments quite similar.

The initial situation was the same for both: we had 5 VMs running, 1 VM for each service. Since these 5 VMs could not hold the high load, it was necessary to acquire further resources. Both cases behaved pretty much the same until minute 20. The first few minutes are very interesting in both case: in case of the *Without Optimization* iteration, the amount of parallel executed workflows was not to high, so that, it was not necessary to acquire more resources for dataloader and pre-processing service. Only the processing service needed more resources.

In the case of the *With Optimization* scenario it is interesting, that ViePEP decided to acquire more resources not only for the first service (dataloader service) but also for the second one (preprocess service). This is explainable by the fact the ViePEP calculated the maximal possible throughput of the first service, which led to the fact, that a higher load for the second service is likely. Therefore, it was necessary to duplicate this service as well. Consequently, the same happened to the third service. Further, as soon as ViePEP realized, that the acquired resources are to much, e.g., the load of some VMs was not high enough, it shutdown the unneeded VMs. In every run, the system found the perfect state after about 30 minutes and did not acquire additional resources anymore.

6.3.2 Linear Arrival Scenario

Figure 6.4 shows the results of the linear arrival scenario experiments, i.e., it shows the average of all 3 runs with optimization and the average of all 3 runs without optimization.

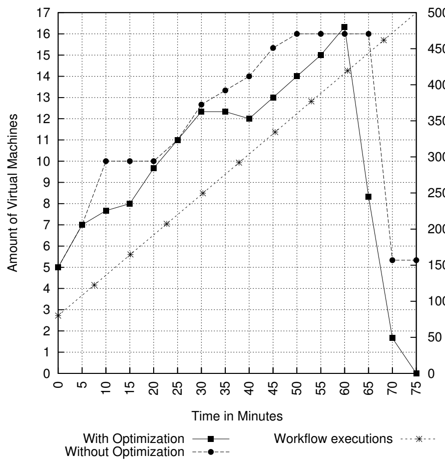


Figure 6.4: Linear Arrival Results (average of all 3 runs)

1. **Without Optimization:** As we can see in the Figure 6.4, it is really obvious that the amount of VMs is increasing while the number of simultaneously executed workflows is increasing. We can affiliate this results to the fact, that a new VM is always started, when the load is getting to high.

   The result in numbers can be found in Table 6.1. In total, this experiment was running in average for about 71.67 minutes and we had at the maximum of 16 VMs simultaneously running. The total costs have been 915 VM-Minutes.

2. **With Optimization:** The results of the linear arrival scenario with optimization can also be found in the Figure 6.4. As we can see, all in all the amount of VMs is increasing linear, but there are some break-ins. These break-ins are explainable because the optimization function is first trying to increase the needed resources according a linear function. But after some time, it realizes that the actual needed amount of needed resources is lower, that is why the unneeded resources are freed again. Because the simultaneously executed workflows is still increasing, the system needs more resources. We can see the rise of active VMs again at minute 20 until all workflows are processed around minute 60. As soon as the queue is empty, the system realizes that the VMs can be removed (at minute 65). The results of this experiment can be found again in Table 6.1.

The results of the linear arrival scenario for the iterations *Without* and *With Optimization* are quite similar to each other. Despite of the fact, that the amount of active VMs are in

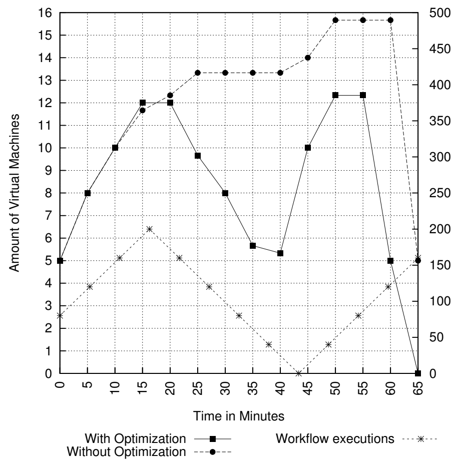


Figure 6.5: Pyramid Arrival Results (average of all 3 runs)

both cases alike, the evaluation with optimization was faster in each case for an average of about 10 minutes. This can be tracked back to the fact, that with optimization, ViePEP acquired exactly as much resources as needed.

In detail: in the case of the evaluation without an optimization a new resource was acquired as soon as the load got to high for a particular service. This means, only a limited amount of workflows could be executed until more resources are available. In the case of running this scenario with an optimization, the needed resources are acquired in advance, which prevented bottlenecks.

6.3.3 Pyramid Arrival Scenario

Figure 6.5 shows the results of the pyramid arrival scenario experiments, i.e., it shows the average of all 3 runs with optimization and the average of all 3 runs without optimization.

1. **Without Optimization:** This scenario was the worst case for our system without optimization, because the amount of parallel running executors are first increasing and then decreasing again. Because of the nature of this unoptimized approaches, no unneeded VM was shut down. Therefore, we literally wasted our resources as the simultaneously running executors has decreased. The results in numbers can be found in Table 6.1.

2. **With Optimization:** The results of the three runs for the pyramid arrival scenario can also be found in Figure 6.5. All in all ViePEP was doing well for the most complicated case. As we can see in the presented figures, the amount of active VMs is increasing in parallel with the amount of running executors. Further, as soon as the demand of needed resources was decreasing, ViePEP decided to shutdown unneeded VMs and reacquired them when they where needed. In this scenario we had an average cost of 576.66 VM-Minutes while the maximum of active VMs was 13.

The pyramid arrival scenario was the worst case for the runs without optimization and big challenge for the runs with optimization turned on. As we mentioned in the description of the evaluation without optimization, new resources are only required when the load got to high, but unneeded resources are not freed anymore. This led literally to a waste of resources, because the system did not react to the decreasing demand of resources. Therefore, the iterations with optimization turned on, produced a much better result, although not a perfect one.

As we can see, in each of the three runs, the system still acquired resources at minute 15 although the amount of parallel running executors was already decreasing. This can be traced back to our optimization approach presented in Section 5.2.3.2. We used linear regression in order to calculate the amount of needed resources for each single service. Since the amount of parallel running executors is increasing in a linear way, the system produces pretty good results for the first few minutes. Since ViePEP can not know, that the amount of needed resources is suddenly decreasing, it still acquires resources as it has calculated. However, as we can see in the Figure 6.5, ViePEP reacts quite fast as soon as the demand is decreasing again and removed unneeded resources. This can be seen around minute 40, when the load is almost 0, we have only one VM for each service (we defined this as the minimum of active VMs, i.e., for each service at least one VM). Further, after the number of parallel running executors is increasing again, ViePEP acquires more resources in order to meet the required load.

6.4 Summary

As we can see in the presented results, our framework performs pretty well under the given evaluation scenarios. Compared to the runs without optimization, ViePEP did not just process the workflow queue faster, but was also cheaper in costs and had a lower error rate. Therefore, we can assume that in these cases, our approach using OLS Linear Regression was the right choice. We can traceback the results of the burst and linear arrival scenario on their linear natures.

As presented in the Table 6.1, the standard deviation for the duration is always below 3. This indicates on the one hand that the measured values tend to be very close to the average and this means on the other hand that our calculations produce very exact results. It further tells us, that the results are reproducible and 3 runs were enough for our evaluation. We can further see, that the the standard deviation with optimization has been always lower than without optimization. Even more, the deviations between the different arrival curves are almost the same. What stands out, is that the deviations for the costs in VM-Minutes is for the pyramid arrival scenario in both cases, with and without optimization lower than in the linear arrival scenario. This can be traced back to the fact, that in the pyramid scenario the maximal amount of concurrent executed workflows is lower than in the linear scenario, and thus, less VMs were needed. The lower VM-Minutes become clear, when we compare the overall execution duration of those two scenarios. The numbers for the lost invocations are also interesting: The high values for the approach without optimization are explainable by the fact, that a VM is only acquired when the load for a single service instance gets too high. This means, it may happen, that too many requests are send to a particular Backend VM and the Backend VM is not able to hold the load anymore, some requests may get lost. In the scenarios with optimization, the values for the lost invocations are almost the same for the burst arrival and the linear arrival scenario, however, the value for the lost invocations for the pyramid arrival scenario are slightly higher.

Nevertheless, although our evaluations have produced pretty good results, we have to admit, that the comparison of our approach against one without optimization is not quite fair. However, to the best of our knowledge, so far no standardized benchmarking system exists for such a BPMS, which could help us to evaluate our work. As we can see in the related work, Han Li et al. [37] and Cardellini et al. [13] did not compare their results against a baseline at all.

In order to evaluate ViePEP more detailed, we could use several different workflows with more different service, to see how our system behaves under different loads. However, this would go beyond the scope of this work, since the evaluations would need much more time and much more resource which we did not had available.

CHAPTER 7

Conclusion and Future Work

Within this thesis, we addressed the concept of Elastic Processes (described in [17] as cost, resource and quality elasticity) in connection with business process management and resource optimization in the Cloud.

Even though Cloud computing has positioned itself as *the* State-of-the-Art technology already a couple of years ago, a lot of research is still going on in this area. Developers, users and researchers are trying to get as much as possible out of the promises coming with Cloud computing. The most famous term in connection with Cloud computing is the word scalability. This is the promise of scaling up/out applications in order to meet the increasing demand of requests or scaling down/in whenever the demand is decreasing.

However, since scalability is only the promise that this is possible, it is still a big challenge to make this possible. Dustdar et al. [17] proposes the term Elastic Processes. The concept of Elastic Processes defines the various facets of elasticity in Cloud computing. The authors name three elasticities: resource elasticity, cost elasticity, and quality elasticity. Since to the best of our knowledge, no BPMS exists which is able to process several hundred of workflows simultaneously while ensuring their SLAs through scheduling their executions and optimizing the provided resources, we created the BPMS ViePEP – the Vienna Platform for Elastic Processes.

ViePEP is able to process hundreds of workflows concurrently while still ensuring the given SLAs. To do so, ViePEP schedules their executions and optimizes the acquired resources while using each VM to its limits. In order to ensure the SLAs defined within the workflows, ViePEP monitors each service invocation in terms of response time and failure rate and further, the underlaying VMs' CPU and Ram. Since ViePEP maintains a queue of waiting workflows it is aware of when, how many workflows have to be processed and consequently how many services have to be invoked. By combining this information with the historical monitored data, ViePEP is able to predict the needed resources for each single service in the near future thus it is time and cost efficiently.

In our evaluation (see Chapter 6) we have shown that our approach produces acceptable results. We have evaluated the same scenario with and without an optimization and the results show, that in each tested scenario, our approach is not only faster, but also cheaper in costs and has a lower error rate (see Table 6.1).

7.1 Future Work

Although ViePEP is in its current state a fully functional prototype, the development is not considered to be finished. ViePEP is designed in a way that every single component is easy extend- or replaceable. Consequently, ViePEP serves as a base for future theses, dissertations or research papers. We plan to extend ViePEP by new features and introduce further improvements to the current implementation:

- Our current workflow description provides sequential workflows, but in a future work we will consider an approved standard, e.g., BPEL [51] or YAWL [69], thus more complex workflows including branches, loops, ... are possible.
- Our proprietary SLA description included in the business process description will be replaced by a standard as well, thus it is more generic and more different SLOs can be defined. As a possible standard for a SLA description, WSLA [42] could be applied.
- The current SLAs definition does not provide a field for penalty costs. Penalty costs apply whenever a SLA has been violated. At the moment, we always try to fulfill the wanted SLAs, although this is not possible in every case, we do not consider any penalties. In addition to penalty costs, other SLOs will be considered, e.g., service response time.
- For now, the threshold which has to be reached, before a new instance is acquired is a fixed predefined value. This has a few drawbacks, e.g., if we have two dif-

ferent service, we estimated the needed resources for the first service at 1% CPU usage for each invocation, in contrast to that, a single invocation for the second service needs about 20% of the CPU. There, it is obvious, that we should fix the threshold for the first service at an higher level than for the second service in order to allow more simultaneously invocations. Since we do not want to predefine this threshold for each service it is likely to make this value dynamic, i.e., by learning from past invocations.

- While we currently only support scale-out and scale-in, i.e., we acquire additional VMs with the same configuration, in a future work we will consider scale-up and scale-down as well, i.e., add additional CPUs or ram to a single VM as mentioned in [44].
- At the moment every Backend VM has the same configuration, i.e., the same CPU and RAM. If we follow a more complex optimization approach, we could make use of different configured VMs. This will lead to a better cost control and the acquired resources can be used more efficiently.
- Our centralized approach for the BPMS could lead to a bottleneck. Therefore, in a future work will will consider distributing our BPMS on several different VMs so that more workflows can be processed simultaneously.
- In a future version of ViePEP we will evaluate different approaches as a replacement for our Linear Regression Model, a possibility would be a self-adaptive prediction model, e.g., Kalman Filters, or $\mathcal{H}_\infty$ Filters.
- In addition to the self-addaptive prediction model, which we will consider in a future work, we will examine more complex optimization approaches which include not only resource optimization but also managing the scheduling plan and the number of concurrent executed workflows.

List of Figures

List of Acronyms

ASP	Application Service Provider	18
BP	Business Process	1
BPMS	Business Process Management System	3
BPM	Business Process Management	1
CaaS	Communication-as-a-Service	17
EP	Elastic Process	3
EPR	EndpointReference	12
HaaS	Hardware-as-a-Service	15
IaaS	Infrastructure-as-a-Service	15
NIST	National Institute of Standards and Technology	2
OLS	Oridinary Least Square	48
PaaS	Platform-as-a-Service	15
QoS	Quality-of-Service	5

RPC	Remote Procedure Call	8
SaaS	Software-as-a-Service	15
SLA	Service Level Agreement	2
SLO	Service Level Objective	13
SOA	Service-Oriented Architecture	6
SOAP	Simple Object Access Protocol	8
SOC	Service-Oriented Computing	5
SWF	Scientific Workflows	1
UDDI	Universal Description, Discovery and Integration	6
VM	Virtual Machine	17
VMs	Virtual Machines	3
WAR	Web Application ARchive	53
WfM	Workflow Manager	37

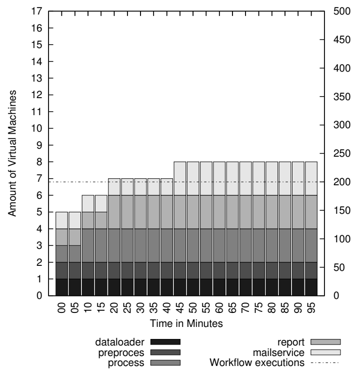
WSLA Web Service Level Agreement 13

WS-BPEL Web Service Business Process Execution Language 11

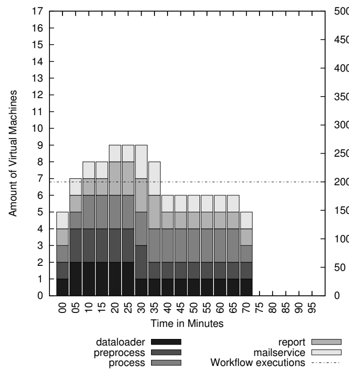
XML Extensible Markup Language 7

APPENDIX A

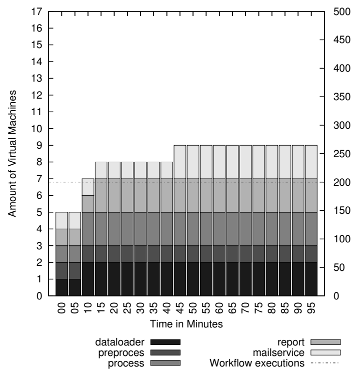
Evaluation Results



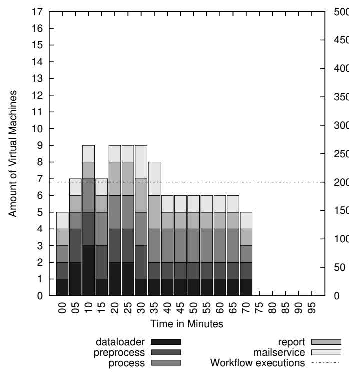
(a) Without Optimization Run 1



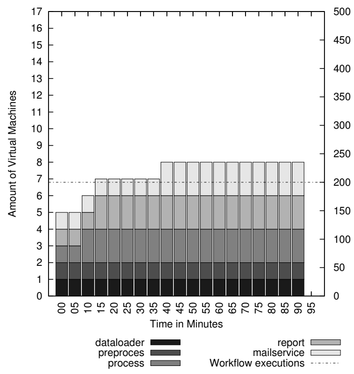
(b) With Optimization Run 1



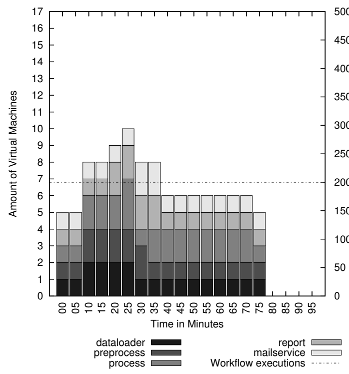
(c) Without Optimization Run 2



(d) With Optimization Run 2

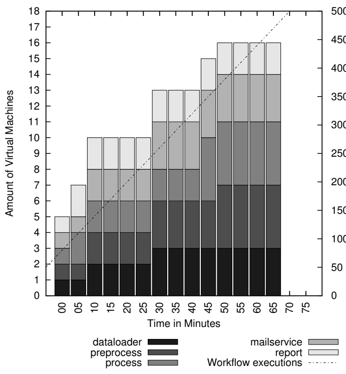


(e) Without Optimization Run 3

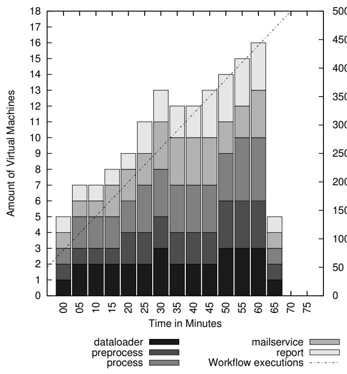


(f) With Optimization Run 3

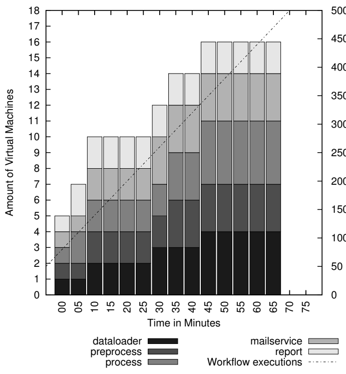
Figure A.1: Burst Arrival Results



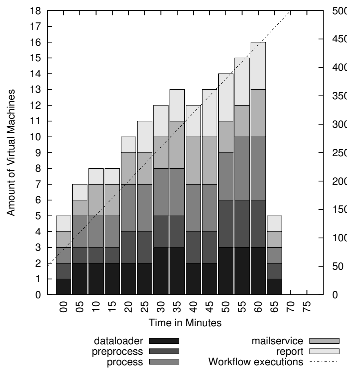
(a) Without Optimization Run 1



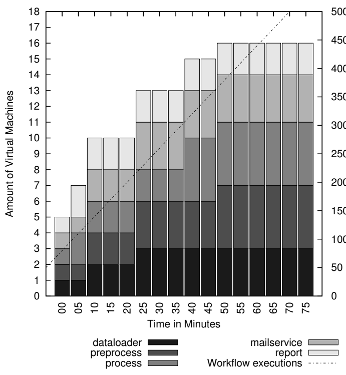
(b) With Optimization Run 1



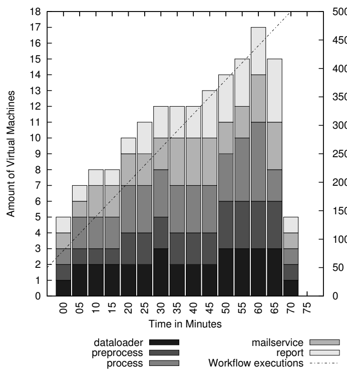
(c) Without Optimization Run 2



(d) With Optimization Run 2

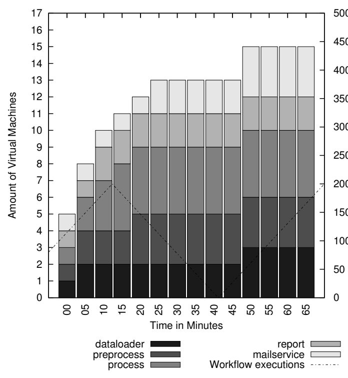


(e) Without Optimization Run 3

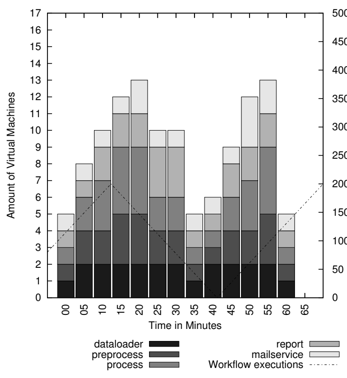


(f) With Optimization Run 3

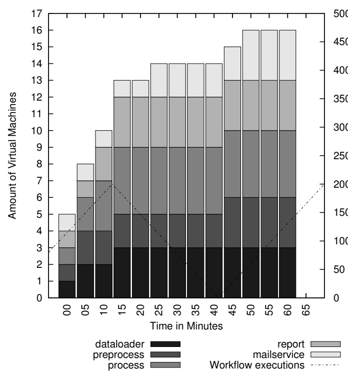
Figure A.2: Linear Arrival Results



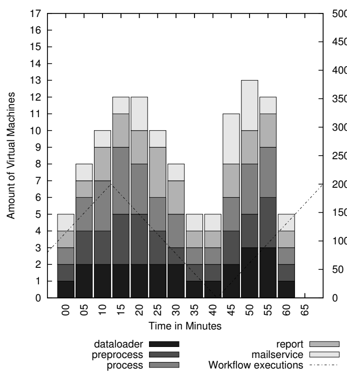
(a) Without Optimization Run 1



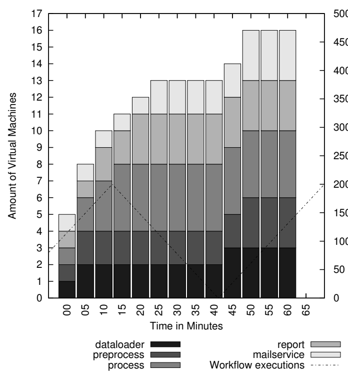
(b) With Optimization Run 1



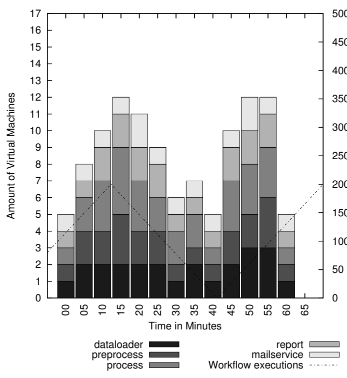
(c) Without Optimization Run 2



(d) With Optimization Run 2



(e) Without Optimization Run 3



(f) With Optimization Run 3

Figure A.3: Pyramid Arrival Results

Bibliography

[1] *Virtualization for dummies*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[2] T. Andrews, F. Curbera, H. Dholakia, and Y. Goland. Business process execution language for web services. May 2003.

[3] Apache. Apache commons math. `http://commons.apache.org/math/`. [Online; accessed September-2012].

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, 2009.

[6] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable cloud services using tosca. *IEEE Internet Computing*, 16(3):80–85, May 2012.

[7] T. Binz, F. Leymann, and D. Schumm. Cmotion: A framework for migration of applications into and between clouds. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1 –4, dec. 2011.

[8] D. Booth and C. K. Liu. Web service description language (wsdl) version 2.0 part 0: Primer. `http://www.w3.org/TR/wsdl20-primer/`. [Online; accessed August-2012].

[9] I. Breskovic, M. Maurer, V. C. Emeakaroha, I. Brandic, and S. Dustdar. Cost-efficient utilization of public sla templates in autonomic cloud markets. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud*

Computing, UCC '11, pages 229–236, Washington, DC, USA, december 2011. IEEE Computer Society.

[10] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[11] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5 –13, sept. 2008.

[12] 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 Gener. Comput. Syst.*, 25(6):599–616, June 2009.

[13] V. Cardellini, E. Casalicchio, F. Lo Presti, and L. Silvestri. Sla-aware resource management for application service providers in the cloud. In *Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications*, NCCA '11, pages 20–27, Washington, DC, USA, 2011. IEEE Computer Society.

[14] T. Charalambous and E. Kalyvianaki. A min-max framework for cpu resource provisioning in virtualized servers using $\mathcal{H}_\infty$ filters. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 3778 –3783, dec. 2010.

[15] W. Chen, X. Qiao, J. Wei, and T. Huang. A profit-aware virtual machine deployment optimization framework for cloud platform providers. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 17 –24, june 2012.

[16] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, Mar. 2002.

[17] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. volume 15, pages 66–71, Piscataway, NJ, USA, Sept. 2011. IEEE Educational Activities Department.

[18] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journals of Web and Grid Services*, 1(1):1–30, August 2005.

[19] Eclipse Foundation. Eclipse bpel project. `http://www.eclipse.org/bpel/`. [Online; accessed August-2012].

[20] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[21] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[22] T. A. S. Foundation. Apache maven 3.0.4. `http://maven.apache.org/download.html`. [Online; accessed June-2012].

[23] R. Garcia and J.-M. Chung. Xaas for xaas: An evolving abstraction of web services for the entrepreneur, developer, and consumer. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 853 –855, aug. 2012.

[24] S. K. Garg, S. Versteeg, and R. Buyya. Smicloud: A framework for comparing and ranking cloud services. *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, (Vm):210–218, Dec. 2011.

[25] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong. The characteristics of cloud computing. In *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, ICPPW '10, pages 275–279, Washington, DC, USA, 2010. IEEE Computer Society.

[26] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to web services architecture. *IBM Syst. J.*, 41(2):170–177, Apr. 2002.

[27] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, Jan. 2005.

[28] L. jie Jin, L. jie Jin, V. Machiraju, V. Machiraju, A. Sahai, and A. Sahai. Analysis on service level agreement of web services. Technical report, HP Laboratories, 2002.

[29] X. Jin and J. Liu. Resource optimization in heterogeneous web environments. *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 46–49, 2005.

[30] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 595–604, New York, NY, USA, 2006. ACM.

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

[32] A. Khiyaita, M. Zbakh, H. El Bakkali, and D. El Kettani. Load balancing cloud computing: State of art. In *Network Security and Systems (JNS2), 2012 National Days of*, pages 106 –109, april 2012.

[33] E. Kühn, R. Mordinyi, M. Lang, and A. Selimovic. Towards zero-delay recovery of agents in production automation systems. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 307–310, Washington, DC, USA, 2009. IEEE Computer Society.

[34] E. Kühn, R. Mordinyi, and C. Schreiber. An extensible space-based coordination approach for modeling complex patterns in large systems,. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 634–648. Springer Berlin Heidelberg, 2009.

[35] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze. Cloud federation. In *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. IARIA, September 2011. Best Paper Award.

[36] P. Leitner, W. Hummer, and S. Dustdar. Cost-based optimization of service compositions. *IEEE Transactions on Services Computing*, 2011.

[37] H. Li and S. Venugopal. Using reinforcement learning for controlling an elastic web application hosting platform. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 205–208, New York, NY, USA, 2011. ACM.

[38] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM.

[39] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, pages 13–18, New York, NY, USA, 2009. ACM.

[40] M. Litoiu, M. Woodside, J. Wong, J. Ng, and G. Iszlai. A business driven cloud optimization architecture. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 380–385, New York, NY, USA, 2010. ACM.

[41] B. Ludäscher, M. Weske, T. Mcphillips, and S. Bowers. Scientific workflows: Business as usual? In *Proceedings of the 7th International Conference on Business Process Management*, BPM '09, pages 31–47, Berlin, Heidelberg, 2009. Springer-Verlag.

[42] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web service level agreement (wsla) language specification. *IBM Corporation*, pages 1–110, 2003.

[43] P. Mell and T. Grance. The nist definition of cloud computing (draft) recommendations of the national institute of standards and technology. 2011.

[44] M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –8, march 2007.

[45] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar. Towards recovering the broken soa triangle: a software engineering perspective. In *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*, IW-SOSWE '07, pages 22–28, New York, NY, USA, 2007. ACM.

[46] B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors, and implications. *Trans. Sys. Man Cyber Part C*, 38(3):280–291, May 2008.

[47] NetBeans Community. Developer guide to the bpel designer. `http://www.netbeans.org/kb/60/soa/bpel-guide.html`. [Online; accessed August-2012].

[48] E. Newcomer and G. Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 2004.

[49] OASIS. Bpel4people. `http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cd-06.pdf`. Accessed: 2013-01-21.

[50] OASIS. Web services business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`. [Online; accessed August-2012].

[51] OASIS. *Business Process Execution Language 2.0 (WS-BPEL 2.0)*, 2007.

[52] OpenStack. Openstack. `http://www.openstack.org/`. [Online; accessed Januery-2013].

[53] Oracle. Sun java jdk 1.6. `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. [Online; accessed June-2012].

[54] Oracle Corporation. Oracle bpel process manager. `http://www.oracle.com/technology/products/ias/bpel/index.html`. [Online; accessed August-2012].

[55] M. P. Papazoglou. *Service -Oriented Computing : Concepts , Characteristics and Directions*. 2003.

[56] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer. Service-oriented computing research roadmap. In *Service Oriented Computing (SOC)*, number 05462 in Dagstuhl Seminar Proceedings, pages 38–45. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[57] R. Rajamony and M. Elnozahy. Measuring client-perceived response times on the www. In *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, USITS'01, pages 16–16, Berkeley, CA, USA, 2001. USENIX Association.

[58] B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM '09, pages 44–51, Washington, DC, USA, 2009. IEEE Computer Society.

[59] S. Rusitschka, K. Eger, and C. Gerdes. Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 483 – 488, oct. 2010.

[60] A. Sahai, A. Durante, and V. Machiraju. Towards automated sla management for web services. *Report HPL-2001-310 (R. 1)*, 310, 2002.

[61] S. Schulte. *Web Service Discovery Based on Semantic Information - Query Formulation and Adaptive Matchmaking*. PhD thesis, TU Darmstadt, September 2010.

[62] J. E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, May 2005.

[63] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[64] A. Tchana, L. Broto, and D. Hagimont. Approaches to cloud computing fault tolerance. In *Computer, Information and Telecommunication Systems (CITS), 2012 International Conference on*, pages 1 –6, may 2012.

[65] UDDI.org. Uddi technical white paper. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`. [Online; accessed August-2012].

[66] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 217 –228, june 2005.

[67] H. N. Van, F. D. Tran, and J.-M. Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[68] H. N. Van, F. D. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02*, CIT '09, pages 357–362, Washington, DC, USA, 2009. IEEE Computer Society.

[69] W. M. P. Van Der Aalst and A. H. M. T. Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.

[70] W. M. P. Van Der Aalst, A. H. M. T. Hofstede, and M. Weske. Business process management: a survey. In *Proceedings of the 2003 international conference on Business process management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[71] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Dec. 2008.

[72] M. Verbeek. *A Guide to Modern Econometrics*. John Wiley & Sons, 2008.

[73] White Paper, Agilent Technologies Inc. Service level agreements - an emerging trend in the internet services market, 1999.

[74] I. H. Witten and H. M. A. Frank, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3 edition, 2011.

[75] World Wide Web Consortium (W3C). Extensible markup language (xml). `http://www.w3.org/XML/`. [Online; accessed August-2012].

[76] World Wide Web Consortium (W3C). Simple object access protocol 1.2 part0(soap). `http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`. [Online; accessed August-2012].

[77] World Wide Web Consortium (W3C). Web service description language (wsdl). `http://www.w3.org/TR/wsdl`. [Online; accessed August-2012].

[78] World Wide Web Consortium (W3C). Xml path language (xpath). `http://www.w3.org/TR/xpath/`. [Online; accessed August-2012].

[79] World Wide Web Consortium (W3C). Xml schema. `http://www.w3.org/XML/Schema`. [Online; accessed August-2012].

[80] J. Yu and R. Buyya. A novel architecture for realizing grid workflow using tuple spaces. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 119–128, Washington, DC, USA, 2004. IEEE Computer Society.

[81] Z. Zhang, H. Wang, L. Xiao, and L. Ruan. A statistical based resource allocation scheme in cloud. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 266–273, Washington, DC, USA, 2011. IEEE Computer Society.