Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2841

Management Framework for Amazon EC2

Frank Bitzer

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dipl.-Inf. Ralph Mietzner Dipl.-Inf. Daniel Wutke
Commenced:	May 29, 2008
Completed:	November 28, 2008
CR-Classification:	C.2.4, D.2.11, K.1

Contents

List of Figures

List of Listings

Chapter 1

Introduction

In the days of steadily growing bandwiths, a computer user is far away from being restricted by the power of his local workstation. He is not even restricted by the power of his company's server farm. Instead, he can use resources from all over the world.

Of course, the same holds not only for a human user, but also and even more for a modern enterprise application. State-of-the-art technologies and architectures like SOA (*Service Oriented Architecture*) are an ideal choice to access heterogeneous resources without having to deal with any kind of compatibility issues or implementation details. These details are hidden behind the standardized interface.

In this context, the umbrella term "*Cloud Computing*" occurs. Sometimes being praised as a revolution, this term mainly means what it says: the user of a software system does not run the software and the required hardware himself, but he obtains the required resources from a supplier, mostly over the Internet. Applications and data are located on a "cloud", scattered over multiple systems that are not really known by the consumer. The term "Cloud" became familar because the Internet often appears as a cloud image in architecture diagrams.

In particular, the user exactly obtains the capabilities he actually needs and of course he also only pays for what he uses. One of the interesting things about Cloud Computing is that this approach enables entirely new types of business to exist and be economically viable that never could have persisted before.

Thinking in this direction, the concept of *Software as a Service* (SaaS), another central topic of these days, can be seen as one of the applications of Cloud Computing. The cloud makes the SaaS model more interesting by reducing the costs associated with producing a SaaS application. Section 2.1.1 provides a classification of these terms amongst other peculiarities of Cloud Computing such as *Platform as a Service*.

There are many ways to exploit the benefits of Cloud Computing. For instance, the power of the cloud can be used to build highly scalable applications as well as highly available systems. Extensive calculations can be outsourced and be executed in the cloud. And there are much more possibilities. Some of them are outlined in Section 1.4 and in Section 2.4.

Currently, a lot of research is done in this sector and many companies are launching new products based on the principle of computing in the cloud. For instance, under the name "*App Engine*"[1] Google provides a service that allows users to run their web applications on Google's own servers. This means these web applications are able to benefit from all the power of Google's infrastructure, including scalability and availability.

IBM is working on plans for "*Blue Cloud*"[2], a series of cloud computing offerings that will allow corporate data centers to operate more like the Internet by enabling computing across a distributed, globally accessible fabric of resources, rather than on local machines or remote server farms.

Among all these service providers, Amazon plays an important role. Providing a whole bunch of services, they seem to be well prepared for the challenges of today. Launched in 2002(!), *Amazon Web Services* (AWS) is a collection of various services ranging from a messaging service (*Amazon SQS*[3]) to a distributed database management system (*Amazon SimpleDB*[4]).

Two other services are worth mentioning, since this work will mainly concentrate on them: *Amazon EC2*[5] and *Amazon S3*[6].

EC2, abbreviation for "Amazon Elastic Compute Cloud", offers infrastructure on demand. It allows you to use resizable compute capacities in the cloud, focussing especially on the aspect of scalability.

Finally, S3 is an online storage service that can be used to manage data on EC2.

1.1 Motivation

With Amazon EC2 and Amazon S3 a developer has two powerful instruments when designing web-scale applications. He can use the Web service interface to allocate an arbitrary number of virtual machines in the cloud, each of them representing a server instance. These virtual machines are based on Linux and can be configured with absolutely any software that is needed. Fully configured servers can be saved on Amazon S3 for further usage and it is possible to launch new server instances within minutes.

There are reams of scenarios where these features might turn useful, ranging from executing simple stress tests on EC2 to building huge distributed applications that scale on demand. In [Var08] Jinesh Varia from Amazon Webservices describes architectures that make use of Internet-accessible on-demand services.

However, the Web service interface that is used to interact with EC2 is only composed of some essential functions. It is not intuitive for an end-user and just doing simple things implies a lot of work.

[1] `http://code.google.com/appengine/`
[2] `http://www-03.ibm.com/press/us/en/photo/22615.wss`
[3] `http://www.amazon.com/Simple-Queue-Service-home-page/b?ie=UTF8&node=13584001`
[4] `http://www.amazon.com/SimpleDB-AWS-Service-Pricing/b?ie=UTF8&node=342335011`
[5] `http://www.amazon.com/gp/browse.html?node=201590011`
[6] `http://www.amazon.com/gp/browse.html?node=16427261`

Second, the tools that are available for administrating EC2 are suffering the same problem and are missing some important functionalities. For instance, it is not possible to do a simple file copy from S3 to an EC2 instance with the same tool that was used to start the instance.

Furthermore, most tools are not web-based. Applications are executed in the cloud, but in order to control the cloud, one is bound to a local machine, where all the necessary tools have to be installed. Managing the cloud from within the cloud would be a reasonable alternative.

And finally, the last point is a consequence of the previous ones: up to now, research and development has mainly covered the basic functions such as starting and stopping server instances or the usage of EC2 in general.

As a conclusion, this thesis goes one step further. It elaborates some enhanced concepts for possible high level functions that sit upon the EC2 API, e.g. a notification mechanism which allows processes to subscribe to events on an EC2 instance.

Another goal of this work is to develop an extensive, web-based management framework for configuring and controlling EC2.

Within the scope of this tool a powerful but easy-to-use service layer will be designed that implements the previously mentioned concepts.
Apart from being usable as simple Java library, the tool will have a rich graphical user inferface for man-machine communication as well as a convenient Web service interface for invoking its functions from other applications.

The presence of a well-designed Web service interface even allows to orchestrate server instances using BPEL processes, a fact that adds a new dimension to the possibilities of EC2, allowing to combine provisioning services offered by EC2 with provisioning services offered by other cloud providers and provisioning engines. For further reading, see Section 1.4 and [ML08]. Section 4.5.3 directly illustrates a use case scenario using a BPEL processs.

1.2 Outline

Mainly, this work consists of two parts. In addition to the work at hand containing the ideas and concepts of the thesis, the full source code for the EC2 management tool including all running examples belongs to this work, too. It can be found on a CD that is delivered seperatly. Furthermore, a website has been build: it can be found at `http://cloud42.net`.

The conceptual part of the thesis comprises five chapters, that can be sketched as follows:

- **Chapter 2 "Fundamentals"** gives an introduction into the world of Cloud Computing and describes the key technologies. The ideas and principles behind Amazon EC2 as well as the tools that are available right now are illustrated and discussed. Finally, some examples are given showing how EC2 can be of use in certain scenarios.
- **Chapter 3 "Enhanced Concepts"** is the core of this work. It elaborates four advanced concepts and features that can or could be developed upon the functionalitiy of EC2.

- **Chapter 4 "Implementation"** describes the implementational part of the thesis. Having summarized the requirements referring to the previous chapter, it deals with the technologies used for implementing the management tool, outlines their pros and cons and briefly introduces the architecture of the system.
- **Chapter 5 "Summary and Outlook"** reviews and concludes the work by commenting the most current developments and the latest news in the world of Cloud Computing.

1.3 Definitions

Since this thesis contains some abbreviations and proper nouns, these are defined in this section in order the provide more clearance for the reader.

The first two terms to be clarified concern the main topic of this work and therefore are essential: **"EC2"** is the name of a service of Amazon allowing users to rent servers on a hourly basic. Chapter 2 will contain a more detailed review of this service.
"S3" is another service from Amazon that is strongly related to EC2, but can be accessed seperatly, too. It is a persistent online file system and can be used to transfer data from and to an EC2 server instance.

There are two more common used terms in the scope of EC2 that should be distinguished:

- a virtual (server) image hosted on EC2 is called **"AMI"**, which stands for Amazon Machine Image
- an actual running server instance is called **"AMI instance"** or simply **"instance"**

As the name says, an AMI is a preconfigured virtual machine image (usually stored on the S3 file system) containing certain pieces of software. By launching it, an AMI becomes an EC2 server instance. The process of creating and saving a new AMI is called **"bundling an AMI"**.
Please see Chapter 2 for a complete explanation how EC2 works; this section only contains the pure definition of terms.

Next, a term related to security may occur in this thesis: **AWS credentials**. AWS credentials are the information that is used by Amazon to identify a particular user's account. Namely, an AWS credential consists of an account id (or user id), an access key and a secret access key.

At last, the EC2 management tool developed in the scope of this work shall be labeled: from now, it will be referred using the name **"Cloud42"**.
Important: Please note that the working title of the project was "ECToo" and therefore this name may still appear in some screenshots or diagrams.

1.4 Related Work

The idea of Cloud Computing is quite new. Even more, this applies for the topic of EC2 in special. As a result, literature is rarely available and mainly concentrates on articles on the Internet or in technical newspapers.

There are some whitepapers published on the web pages of Amazon. For instance, in [Jon07], de Jonge, a writer for IBM developerWorks, explains how Amazon EC2 can be used to deploy distributed J2EE applications.

In a publication discussing the benefits of "Cloud Computing for large-scaled data-intensive applications" ([LO08]), the authors Liu and Orban introduce a programming model called GridBatch, "which aims at solving large-scale data-intensive batch problems under the Cloud infrastructure constraints." According to their article, "GridBatch is a programming model and associated library that hides the complexity of parallel programming, yet it gives the users complete control on how data are partitioned and how computation is distributed so that applications can have the highest performance possible." Furthermore, they show a running example using Amazon's Computing Cloud.
Liu and Orban focus on hiding the complexity of the cloud whereas this thesis actually tries to develop some concepts not to hide, but to manage and control this complexity on a high level, so that other applications can take advantage of it.

Garfinkel from Harvard University examined the quality of the services offered by Amazon, "because the quality of a single commercial Internet service can literally affect hundreds of millions of individuals" [Gar07]. He found that it fullfills its requirements and it is ready to use, even though the lack of a Service Level Agreement (SLA)[7].
This examination shows that EC2 and its related services build a solid platform for computing in the cloud and that any further research is totally justified.

There is another article covering EC2 in which Mietzner and Leymann sketch a way how the cloud could be of use for SaaS providers by presenting a "unified provisioning infrastructure for SaaS applications" [ML08]. The key idea is that SaaS providers can aquire new servers on demand when the limits of their own data centers are reached. Technically, they use Web services and BPEL processes to aggregate vendor specific provisioning services (such as installation scripts), so that they can be orchestrated and accessed at a higher level.
This article is strongly related to this thesis, since the work at hand evaluates such concepts of abstraction especially for EC2 and the resulting tool Cloud42 will be a provisioning service for Amazon's cloud, providing its methods through a Web service interface that can be easily invoked from BPEL processes.

A rather practically orientated introduction into Cloud Computing including an analysis of some of the currently available services can be found in [Mil08]. Moreover, [Mur08] provides a deep insight into EC2 and its related services from a developer's point of view.

Apart from theoretical work, there are some tools available that can be used to interact with EC2 without having to use the rudimental Web service interface provided by Amazon.

[7] Update: Since October 23, 2008, AWS provides a SLA guaranteeing 99.95% availability for EC2. See `http://aws.amazon.com/ec2-sla/`

Some of them are commercial like *RighScale*[8], a service that can be used to control huge clusters of Amazon server instances.

A complete overview over the existing tools and technologies can be found in the next chapter.

[8] `http://www.rightscale.com/m/`

Chapter 2

Fundamentals

Having given a short overview in the introduction, this chapter revisits the key technologies and services related to this work. Its main purpose is to lay the cornerstones for the next chapter, where the base principles are enhanced by own concepts.

2.1 Cloud Computing and its Subtopics

There is neither a formal definition of *Cloud Computing* in literature nor a strict differentiation to its related concepts.

Cloud Computing can be seen as a computing paradigm where tasks are accomplished by a collection of services that are connected over a network, mostly the Internet.

In a study discussing whether Cloud Computing is ready for the enterprise, Forrester Research recently inverviewed 30 top IT companies and stated the cloud as "[a] pool of abstracted, highly scalable, and managed compute infrastructure capable of hosting end-customer applications and billed by consumption." [FR08]. They came to the conclusion that Cloud Computing does not conform to the needs of the enterprises yet, mainly because it has not emerged the early-adopter phase for now.

For sure, Cloud Computing has its origin in *Grid Computing*, where multiple resources are aggregated to a powerful system that can be accessed as a whole.

However, thanks to modern technologies and mainly inspired by SOA, Cloud Computing goes one step further. The "cloud" can be spread over multiple data centers and server farms. Actually, the end user does not know or does not care where exactly his request is executed. The main thing is, he gets a response. Usually, a Web service interface is the common way to interact with the cloud.

Another fact that differs Cloud Computing from Grid Computing is the scalability of the cloud. To say it metaphorically, the cloud never ends. From a user's point of view, he can delegate as much work to the cloud as he wants, whereat the cloud always provides the same response times. Internally, the cloud dynamically grows and shrinks by acquiring exactly as much resources as needed to fullfill its requirements. For this reason, Cloud Computing is often called "on-demand computing". Usually, a

user is only billed for the resources (calculated in CPU load, memory load or just execution time) that he actually uses.

So the new thing about Cloud Computing is less its capability to provide functionality over the Internet or to offer external computing power. Actually, this can be done since years by using various web applications or traditional data centers. It is rather its internal structure that makes it quite revolutionary. Used the right way, Cloud Computing allows to develop highly scalable and highly available high-performance applications.

In order to achieve this goal, the underlying infrastructure must make use of some state-of-the-art technologies, that mostly are:

- *virtualization*: virtualization is a complex topic, but in connection with Cloud Computing, it is used to decouple software from hardware. This way, multiple applications or even multiple operating systems can be run on one actual computer. Contrariwise, multiple computers can be used to run one application.
- *cluster computing*: especially in conjunction with the above mentioned virtualization, a cluster is needed to serve the needed processing power. By using modern network connections, the cluster can be connected over the Internet without suffering drawbacks concerning performance.
- *service-orientation*: in order to abstract from concrete implementations and to achieve loose coupling (e.g. to overcome failures of single components), the system should allow to compose applications using discrete services.
- *policies*: policies are used to guarantee special Service Level Agreements (SLAs) that define non-functional properties such as maximal response times. Also, SLAs are used to automatically manage the system. If the system experiences peaks in load, it must be able to acquire new resources, so that it is capable of fullfilling its requirements again.
- *scalability*: this is a base requirement. For the system as a whole being scalable, more servers must be able to do more work. Ideally, a linear scalability is desired, since this makes growing the application really predictable and efficient.
- *failover*: in order to work without disruption and loss of data and to provide a high availability and reliability, the system must implement failover technologies. Again, policies are used to arrange a certain quality of service.
- *data management*: since the cloud has to deal with lots of data, efficient data management is essential. Data must be partitioned, fetched from and distributed to anywhere. New technologies like modern online file systems are the first choice.

Taken by oneself, none of these technologies is completly new, but it is not a long time since big companies like Google or Amazon have started composing them to what means the cloud.

In a way, the cloud can be seen as a new kind of middleware that enables entirely new business models.

2.1.1 Software as a Service

One of these business models is *Software as a Service* (SaaS).

Instead of buying a traditional licence and hosting a software on their own system, SaaS users "buy a subscription from the publisher" [Cho07].

In order to be able to sell his service to a wide range of customers, a SaaS provider has to have a powerful infrastructure at his disposal. He must be able so serve different customers consuming different variants of his service. This raises new challenges like the need for multi-tenancy architectures. [KNL08] shows the usage of various multi-tenancy concepts for SaaS applications.

Of course, the cloud is an ideal choice to meet these conditions.

Because of this, Cloud Computing is often mentioned in combination with Software as a Service. However, the term of Cloud Computing is more general and SaaS actually is just one peculiarity amongst others, as discussed in [Lan08a] and [Lan08b]. The cloud is the means, SaaS may be the end.

2.1.2 Infrastructure as a Service

Under the umbrella term "IaaS" that stands for *Infrastructure as a Service*, a new taste of Cloud Computing has emerged. An IaaS cloud provides "resources such as servers, connections, storage, and related tools necessary to build an application environment from scratch on-demand" [Bar08].

This way, IaaS is in some kind the base for Cloud Computing, since it builds the underlying infrastructure. It is not only possible to run software in the cloud, but you can even use it to acquire hardware resources.

In doing so, some new opportunities turn up. Now everyone is able to build (and also sale) cloud applications by himself by dynamically hiring resources in the cloud. Handling IaaS is quite more difficult than just using software in the cloud, but it enables a great flexibility.

Typically, virtualization is used by IaaS providers to open parts of their infrastructure to their clients.

Amazon with its Amazon Web Services[1] products can be seen as the leading IaaS provider, offering various kinds of resources ranging from online storage (S3) to whole virtual servers (EC2). Because of this, Amazon Web Services and especially EC2 will be on the main focus of this thesis.

[1] `http://aws.amazon.com`

2.1.3 Platform as a Service

There is another mentionable subsection of Cloud Computing: *Platform as a Service* (PaaS).

PaaS providers offer readily configured software platforms in the cloud. Developers can use these platforms to deploy their applications, benefiting from the scalability and the other advantages of the cloud. Furthermore, they don't have to care for system administration etc. by themselves. An example for such a platform could be a complete Python application stack as offered by Google App Engine[2].

Often, PaaS clouds are designed within IaaS clouds, but they are more convenient to use, of course by lacking lots of flexibility, because the user is bound to a particular stack of technologies.

For further reading, [Law08] provides a paper on developing software with PaaS technology.

2.2 Amazon EC2

As mentioned above, Amazon is the leading IaaS provider. Under the name "Amazon Web Services" they offer a lot of services, amongst which EC2 probably is the most meaningfull and the most powerful representant.

According to Amazon's web page[3], "Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud.".

The compute capacity is represented in form of fully configurable virtual server instances.

This way, EC2 provides absolutely full flexibility by allowing the user to install or to deploy whatever application he wants to his server instance(s).

Instances can be accessed via SSH or even using VNC or similar tools, if the instance has a graphical desktop.

Configurations can be saved on S3 as *Amazon Machine Images* (AMIs, see 1.3). It is possible to configure several AMIs and to start various instances for different purposes.

Originally, all AMIs were supposed to run a Linux operating system. Meanwhile, there are Windows-based AMIs available, too.

There is one drawback: an EC2 instance itself has no persistent storage system. This means, as long as the instance is running, the user can read and write data from its virtual hard disk. However, as soon as the instance is terminated, all data is lost. But Amazon is working on a persistent storage system for EC2 at the moment, so this is expected to change[4].

There are different instance types available. Since the customer hires a virtual server and does not purchase or lease a particular processor that is used for months or for years, the computing power is calculated in *EC2 Compute Units* (ECU), that change with time according to technical progress.

[2] `http://code.google.com/appengine/`

[3] `http://aws.amazon.com/ec2`

[4] Update: since August 21, 2008, a service called *Amazon Elastic Block Store* (EBS) offers persistent storage for Amazon EC2 instances

Currently (in mid 2008), one ECU equals the power of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

The smallest instance type has one ECU, the largest one reaches a power of 20 ECUs. The single instance types are not listed here, because you can easily obtain their descriptions from the website[5]. Same holds for pricing. The fees for using EC2 depend on the instance type and the time the instance is used. So the pricing model is very characteristic of Cloud Computing and conforms well with the "pay only for what you use" principle.

Of course, the technology used for EC2 and the other services of AWS was not developed from scratch. Instead, Amazon used its own internal infrastructure and technologies and made them available to developers. The virtual machine images rented to the users are hosted in the cloud, or to be more accurate, on the grids of Amazon's data centers all over the world.

The service EC2 is designed to be used in conjunction with other Amazon Web Services. For instance, S3, an online file system, can be used to overcome the above mentioned lack of a persistent storage system. Furthermore, Amazon SimpleDB (a database system) or Amazon SQS (a message queue) can be used to build multi-tier business applications.

Two features of EC2 helping to develop failure resilient applications are worth mentioning.

First, it is possible to run instances in different *Availability Zones*. Availability Zones are distinct locations that are logically seperated from each other. This way, failures in one Availabilty Zone can be bridged by simply switching to an instance in another zone. Of course, the user has to implement mechanisms like Hot Backup himself in order to make use of this feature.

Second, a technology called *Elastic IP Addresses*, that was designed especially for Cloud Computing, lets users associate an IP address with their EC2 account in general instead of with a particular server instance. The user has complete control and can decide to which instance the address actually should be mapped. When one instance fails, it is possible to re-assign the Elastic IP Address to another instance within seconds without having to wait for DNS to propagate the new address.

2.3 Tooling

This section reviews the most common tools and utilities that can be used to configure and to control Amazon's cloud. There are tools provided both by Amazon itself and by third party vendors. Some utilities are free, others are subject to charges. Both developer tools such as libraries and uilities rather adressed to end-users or administrators like graphical management consoles are covered. Of course, there are some other tools not mentioned here, but they do not play such an important role.

In order to achieve some lucidity, all tools are shortly described at first and then estimated by using the following criteria:

- Functionality
- Documentation

[5] `http://www.amazon.com/Instances-EC2-AWS/b?ie=UTF8&node=370375011`

- Convenience

The first point clarifies what you can do with the particular tool, whereas the second and third criteria rather examine how you can do that and how easy it is to embed its functionality in your own applications (if possible, of course).

2.3.1 Amazon EC2 Web Service Interface

The Web service interface offered by Amazon itself allows controlling EC2 by using a stateless Web service. Amazon provides a simple WSDL file[6] that can be used as specification and to build proxies from. The API Version 2008-02-01 was examined in this comparison, but new and improved versions are published frequently.

Functionality

The Web service interface offers methods for accomplishing all required base tasks related to controlling instances.

It provides methods to register AMIs as instances, to control instances by starting, stopping and rebooting them. Furthermore, you can create, edit and modify security groups and RSA keypairs that are used to access running instances over SSH.

Handling Elastic IP Addresses is also supported.

There is also a tool called "Javascript Scratchpad for Amazon EC2" that can be obtained from Amazon's web page and that can be used to explore the Amazon EC2 API by building sample requests and retrieving sample responses.

Documentation

Unfortunately, only the plain WSDL file is provided without any further documentation.

The function of most methods is inferable from their names, but there are also some parameters whose usage is not clear for the user.

[6] `http://s3.amazonaws.com/ec2-downloads/ec2.wsdl`

Convenience

Basically, using a Web service interface is the ideal choice when invoking another application's functionality from an own application.

However, in this case there are two drawbacks that complicate the situation. At first, as mentioned above, the documentation is very poor. Second, the interface is very rudimental. Although it supports all required operations, it is quite cumbersome to work with it, since it gets very granular and does not offer some convenience functions.

Moreover, extended functions that might be needed very often (e.g. like copying files to an instance) are not in the scope of the Web service interface and must be accomplished using other tools.

2.3.2 Amazon EC2 Command-Line Tools

The *EC2 Command-Line Tools*[7] are a wrapper around the EC2 Web Service Interface. As the name says, they are installed locally and provide access to EC2's functions using a command line interface. The version based on the EC2 API Version 2008-02-01 was reviewed and used throughout this work.

Functionality

Being a wrapper for the Web service API, the functionality is exactly the same and concentrates on the base functions.

Documentation

In contrast to the Web service interface, the EC2 Command-Line Tools are documented very well on the web pages of Amazon[8]. There are running examples for each command as well as an extensive "Getting Started" guide[9] that explains the first steps in order to run an instance.

Convenience

Including command line calls into an own application is not the best way of putting things together. Even if you do so, the lack of convenience functions or extended functions as discussed above remains.

[7] `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351`
[8] `http://docs.amazonwebservices.com/AWSEC2/2008-02-01/DeveloperGuide/`
[9] `http://docs.amazonwebservices.com/AWSEC2/2008-02-01/GettingStartedGuide/`

2.3.3 Amazon EC2 AMI Tools

The third toolset offered by Amazon are the *EC2 AMI Tools*[10]. They are intended to be executed on a running AMI instance and therefore differ a bit from the tools reviewed by now. Again, this work deals with the version based on the EC2 API Version 2008-02-01.

Functionality

With the EC2 AMI Tools a user can bundle new AMIs from existing ones and save them to S3. "Bundling an AMI" means creating and persisting a new virtual machine image representing the actual configuration.

Documentation

Like the EC2 Command-Line Tools, the EC2 AMI Tools are described elaborately at the online resource center[11] on Amazon's web pages.

Convenience

Already being installed on EC2's default AMIs, the EC2 AMI Tools can be accessed via command line on the instance, connected via SSH. Unfortunately, at the moment there is now way to invoke tasks like bunding AMIs via a Web service interface.

2.3.4 Typica

The first tool to be reviewed that is not published by Amazon itself is the *Typica* Java library. The project website[12], hosted on Google Code, states Typica as "A Java client library for Amazon's SQS, EC2, SimpleDB and DevPay LS web services".

After all, Typica is an Open Source Java wrapper around the EC2 Web service API, that was developed in order to make accessing Amazon's Web Services more convenient for Java developers. It does not only support EC2, but also SQS and some other services.

While this work was was written, Typica 1.3 was the current version of Typica and therefore builds the base for both this comparison and the Cloud42 implementation.

[10] `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368&categoryID=88`
[11] `http://aws.amazon.com/documentation/`
[12] `http://code.google.com/p/typica/`

Functionality

In this work, we will concentrate on the EC2 capabilities of Typica.

Being built as a Java proxy based on the WSDL provided by Amazon, the functionalities mainly are equitable with those of Amazon's Web service interface.

Although the developers of Typica added some convenience functions, the library only suits for accessing Amazon's API at a quite low, fine grained level. However, this purpose is served very well. Typica is said to work reliable und is used by a number of projects[13].

Documentation

A sore point with Typica is its poor documentation. There is a small code example[14] on the project page. And there is a JavaDoc, but it mainly consists of auto-generated method descriptions. Sometimes, it is difficult to explore the meaning of particluar function parameters, since they are just nominated `arg0`, `arg1`, ...

Of course it is possible to have a look at the source code to see how the parameters are passed to the EC2 API, but it is a quite cumbersome work.

Convenience

In the end most of Typica's functions are obvious and understandable despite the lack of documentation.

By providing direct Java support, using Typica is a good deal for anyone who needs to access the Amazon API at a low level from within a Java application. There is no more need to create an own proxy based on Amazon's WSDL for Java developers.

Furthermore, using a reliable library is definitly a better choice than having to implement various command line calls as required when using the Amazon EC2 Command-Line Tools. This way, there is no need to install the Command-Line Tools on the target machine and you are absolutely platform-independent.

2.3.5 ElasticFox

ElasticFox[15] is a Mozilla Firefox extension that provides a graphical user interface to manage EC2. So, in contrast to the tools reviewed by now, it is not only interesting for developers, but also for administrators who have to manage their pool of EC2 AMIs and instances.

Furthermore, the source code of ElasticFox demonstrates how to use the EC2 API from JavaScript.

[13] `http://code.google.com/p/typica/wiki/TypicaInTheWild`
[14] `http://code.google.com/p/typica/wiki/TypicaSampleCode`
[15] `http://sourceforge.net/projects/elasticfox/`

Version 1.5 was regarded for this section.

Functionality

Again, ElasticFox is a tool that mainly covers the base functionalities of EC2 like starting, stopping and rebooting instances, configuring security groups and managing RSA keypairs.

In order to accomplish these jobs, it is suited well, since it provides a well arranged user interface.

However, its limitations are reached quite early. As soon as a user wants to do things like copying files between instances or other actions that could be considered as trivial, additional tools are needed.

Another disadvantage of ElasticFox is the fact that it is not web based. So it is only possible to manage instances from the local machine, where ElasticFox is installed and the credentials and keypairs are stored.

Documentation

Since ElasticFox is a management tool that does not have an interface allowing to invoke its functionalities from other applications and since the user interface of ElasticFox is self-explanatory, there is no need for a seperate documentation.

Convenience

Working with ElasticFox is easy. Same holds for its installation as simple FireFox plugin.

2.3.6 Rightscale

Rightscale [16] represents a commercial contractor offering lots of services around EC2. None of their tools and services was tested in the scope of this work. However, Rightscale is worth mentioning, because they are the leading service provider around EC2 and a good example not only of what is possible today, but of what could be the rule in future, too.

The functionality of Rightscale was evaluated in July 2008 using a free demo account and by inspecting online documentations.

[16] `http://www.rightscale.com`

Functionality

On their website[17], they solicit a lot of features, ranging from a dashboard that helps to manage AMI's and instances to a tool to enable and control automatic scaling in an easy way. Furthermore, they offer a tool to manage MySQL Servers on EC2/S3 and some features related to load balancing. The portofolio is completed by a lot of predefined configuration scripts for different kinds of AMIs.

A short review of the free online demo stated that Rightscale's "Dashboard" is much more powerful than ElasticFox. For instance, they integrated useful features such as a S3 file browser and the possibility to bundle new AMIs with a few mouse clicks.

Documentation

Rightscale is a commercial contractor and therefore documentation and support is suspected to be good.

Convenience

Due to the fact that testing the services would have gone beyond the scope of this thesis, a statement is not possible here. However, the website and available online examples give a good impression.

2.4 Examples of Use

As we have already seen in some examples throughout this work, the cloud in general and EC2 in special can be used to serve many purposes. The probably most important fields of application are:

- Stress tests
  The power of the cloud can be used to test the performance of an application.
- Scientific calculations
  An aspect that was not mentioned yet is that Cloud Computing also has a deep impact on science. Up to now, an engineer or a scientist needed a huge computing grid when he wanted to do extensive calculations. Usually, only big institutions were able to afford such grids. Now, with EC2 there is a profitable alternative. By running dozens of the most powerful instances EC2 has to offer, enough power for most calculations can be acquired. As an example, even 100 instance hours of the High-CPU Extra Large instance type merely cost 80 $.
- Scalable applications
  By launching additional instances when required, high scalabilty can be reached. This topic was addressed several times in this thesis. For further reading, also see [Fro08] and [OC08].

[17] `http://www.rightscale.com/m/features.html#1`

- Highly available applications
  Techniques like Hot Backup can be used to quickly switch to another instance, when one instance is broken for any reason.
- On-demand web hosting
  Finally, the virtual server instances can be seen and rented as usual web servers. This way, traditional web hosting is possible with the difference that only the resources actually required are billed. Depending on the actual scenario, this could be cheaper than ordering dedicated servers, especially when the full computing power is only needed during short peak times.

By significantly reducing the amount of investment needed for setting up a sustainable infrastructure, EC2 also offers new possibilities for Internet startups. [Nic08] provides a good introduction in the means of the cloud.

For instance, there are companies like iLike[18] that have gone from zero to millions of users in the space of few months or even weeks. Without EC2, they would have spent most of their time buying and installing new servers in order to meet the growing demand, an undertaking which is not only costly, but also time-consuming. But thanks to the cloud, such startups can use parts of the infrastructure of other, etablished companies to go on with their success stories.

[18] `http://ilike.com`

Chapter 3

Enhanced Concepts

Apart from listing images, starting and stopping instances and other fundamental tasks, there are a lot of advanced functions that are desirable for a comprehensive management tool for EC2.

This chapter elaborates some of such enhanced features, illustrates solutions and discusses the pros and cons of different approaches when realizing them.

Although the following part of the thesis is quite abstract and does not cover implementational details, most of the highlighted features are put into practice by integrating them into the Cloud42 management tool that was developed during this thesis.

3.1 Controlling Instances

In order to reach full flexibility and as a precondition for many other features, it is desirable to be able to remotely execute arbitrary commands on a running AMI instance. This way, a user or a BPEL process can interact with the server instance using Cloud42's interface as he/she or it would be connected to the server using a remote shell.

3.1.1 Architecture

As shown in Figure 3.1 and Figure 3.2, there are basically two approaches how communication can be organized.

Figure 3.1 illustrates an architecture where Cloud42 serves as an intermediary between a client process and the AMI instance. The user or process passes commands to the API of Cloud42, which is responsible for interaction with the actual AMI instance.

Typically, a SSH connection is used between Cloud42 and the instance, since the usage of SSH is very common for Amazon's Web Services in general and does not require any configurations or installations on the AMI.

On the other side, in order to be accessible from for instance a BPEL process, Cloud42 exposes its Remoting API as a Web service.

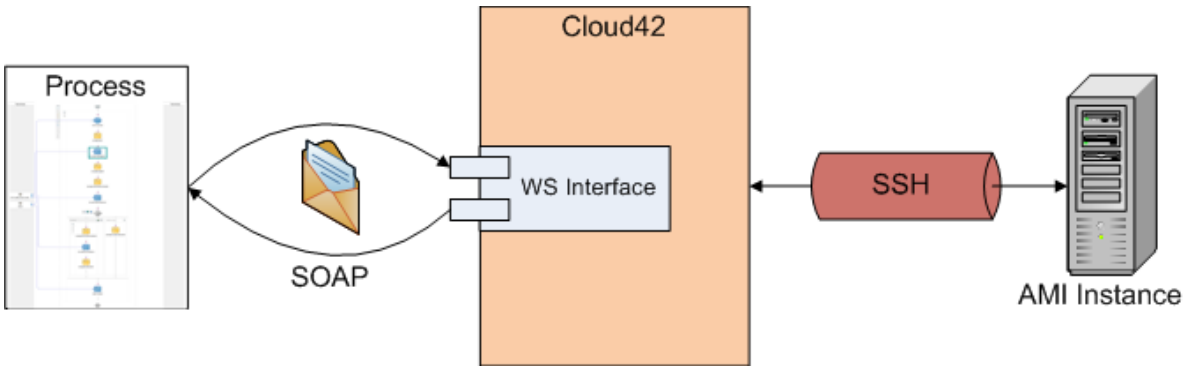

Figure 3.1: Remoting - architecture

Apart from this approach, Figure 3.2 shows an alternative. The indirection over Cloud42 is omitted by allowing the client process to directly communicate with the AMI instance.

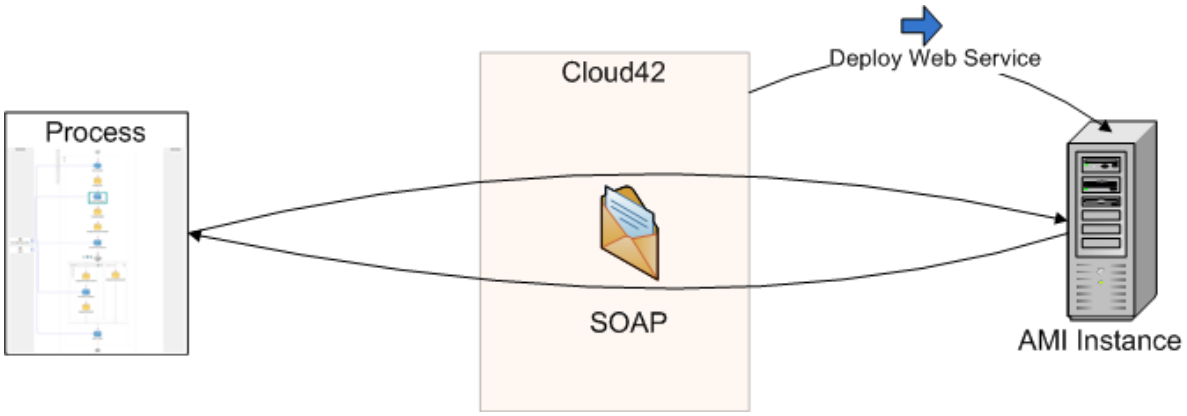

Figure 3.2: Remoting - alternative approach

For sure, this has the advantage of faster response times since data is transfered only once and directly returned to the final destination, namely the client process.

But there are some disadvantages too, amongst which the most important one is the following: the AMI itself must expose an interface that allows to interact with it. In order to reach a loose coupling and to support calls from within BPEL processes, this should be a Web service interface and using SSH is not an option. However, one of the key ideas behind Cloud42 is the fact that it should not require any specific configuration on the AMI or even be restricted to special, pre-configured AMIs. Obviously, the approach illustrated in Figure 3.2 contradicts this idea, since it requires the presence of an extensive Web service layer on the AMI. And in order to be able to deploy these services, additional infrastructure like application servers and maybe even databases are needed.

Of course, Cloud42 could be instructed to install all required elements and to deploy the Web service on an AMI instance as soon as it was started. But this procedure costs a lot of resources so that the savings in response times are caught up by far. Furthermore, related problems occur: what if the instance was

not started by Cloud42 itself, but by a third-party tool? How to make sure that all prerequisites are installed before a client process tries to access the interface for the first time? Solving these problems would require the elaboration of complicated mechanisms, most of them leading to the need to invoke some logic of Cloud42 nearly every time before accessing an AMI directly.

Finally, these cons outweigh the pros and therefore the first solution (see Figure 3.1) was chosen.

3.1.2 Returning Output

Another topic related to executing commands are return values. When a command is sent to and executed on an AMI instance, it may run successfully, maybe returning some output, or cause an error, for instance if it was an invalid command or a file was not found. There is even a third possibility: an exception occurs sending the command, because of connection problems, wrong authentification information or other problems.

Unfortunately, it is not possible to determine definitely whether an arbitrary command failed on the AMI instance or not because there are a variety of different commands and applications that might be invoked. Some of them return proper exit codes, some do not. Some of them raise error messages, some do not. Some of them even write status information to the error output if they returned successfully (for instance, *Curl* does so).

However, getting the output is essential for the process which fired the command as part of a workflow, since it might serve as input for the next operation. Furthermore, retrieving failure messages is at least as important in order to be able to initiate error handling.

Because of this, Cloud42 introduces a concept allowing to work around these issues on client side. All remote operations return so-called `RemoteResult` objects that exactly contain the four kinds of output information mentioned above: the exit code, a possible exception message as well as the error output and the standard output of the command. This way, the caller always gets the full information and is able to evaluate the results by itself, depending on the actual command that was sent.

Listing 3.1 shows an example request for the Cloud42 Web service interface, telling the system to execute the command "echo hello".

Listing 3.1 Example request to execute a command on an AMI

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:web="http://webservice.cloud42.jw.de">
   <soapenv:Header/>
   <soapenv:Body>
      <web:executeCommand>
         <web:dnsName>ec2-67-202-53-93.compute-1.amazonaws.com</web:dnsName>
         <web:rsaKey>-----BEGIN RSA PRIVATE KEY----- ... </web:rsaKey>
         <web:command>echo hello</web:command>
      </web:executeCommand>
   </soapenv:Body>
</soapenv:Envelope>
```

The corresponding result is presented in Listing 3.2. In this case, no exception occurred and the command was run successfully, returning no error message, but the String "hello" on the standard output.

Listing 3.2 Output as `RemoteRessult`

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns:executeCommandResponse xmlns:ns="http://webservice.cloud42.jw.de">
         <ns:return type="de.jw.cloud42.core.domain.RemoteResult"
             xmlns:ax210="http://domain.core.cloud42.jw.de/xsd">
            <ax210:exceptionMessage xsi:nil="true"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
            <ax210:stdErr/>
            <ax210:stdOut>hello</ax210:stdOut>
            <ax210:exitCode>0</ax210:exitCode>
         </ns:return>
      </ns:executeCommandResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

3.1.3 Authentification

In order to communicate via SSH, authorization is required. Amazon uses RSA keypairs to control access to AMI instances. When starting a new instance, the name of the keypair to use can be passed as a parameter to the EC2 API.

When executing remote commands over SSH, Cloud42 itself must be able to login at the AMI instance, too. Therefore, it has to know the user's private key for the particular instance.

In general, there are two possibilities to maintain these private keys.

The first alternative is to provide them to the Cloud42 API with each request. Cloud42 passes them through and uses them for SSH authentification.

A second possibility is to store the keys within Cloud42. This way, it is not neccessary to transfer the key itself with each request. Instead, a handle, most suitable the name of the key, can be passed to Cloud42.

For sure, this reduces the amount of data to be sent over network as well as the risk that someone gets hold of the key who should not know it by intercepting the SOAP messages.

However, in order to "register" it on Cloud42, the key must be sent at least one time anyway, a fact that relativizes the advantage mentioned above, since this registration message could be intercepted, too.

Furthermore, maintaining keys within Cloud42 requires state management and of course some kind of persistency, for instance a database. This does not fit the principle of the Cloud42 Web service layer that is designed as stateless in order to provide a maximum of flexibility and ease of configuration.

After having weighed these two approaches against each other, sending the key in each request was considered to be the better solution.

3.2 File Transfer

Of course, copying files from and to a running server is essential, whether it is a traditional server or an AMI instance. Therefore, file transfer capabilities are an important feature of Cloud42.

3.2.1 Local Files

The first and most obvious file transfer capability is to copy files from a local storage volume to an AMI instance. Furthermore, retrieving files from the AMI must be possible, too.

According to the architecture presented in Section 3.1.1 and illustrated in Figure 3.1, a client process must be able to send files to the Web service API of Cloud42 as well as to retrieve downloaded files from it. Internally, Cloud42 uses the *Secure Copy Protocol* (SCP) over a SSH connection to transfer the files to and from the AMI instance.

Apart from embedding binary data into the SOAP message directly by encoding it as base64 string, there are some more efficient ways to transfer such data. The most popular mechanisms are *SOAP Messages with Attachments* (SwA) ([W3C00]) and *SOAP Message Transmission Optimization Mechanism* (MTOM) ([W3C05a]).

SwA and MTOM

Following similar concepts, both approaches are based on the idea of excluding the binary data from the SOAP envelope itself.

Instead, files are added as attachment in their native format within a multipart MIME message, thus significantly reducing the amount of data to be sent over network, because data can be transmitted as raw bytes. The SOAP message part simply contains references to the other parts of the message.

However, the SwA approach leads to a problem in general: the attachment is no longer part of the SOAP message itself, there are two data models instead. The attachment can be compared to an attachment to an email: the message may reference it, but there are no mechanisms making sure whether it really exists or to encrypt it with the message. As a result of not actually being part of the XML SOAP message, common WS-* mechanisms like WS-Security can not be applied to the attachment.

MTOM overcomes this constraint by leveraging existing standards for securing information such as WS-Encryption and WS-Signatures without requiring specialized standards for securing attachments.

To reference the binary attachments of the message, MTOM uses the XOP:include element defined in the *XML-binary Optimized Packaging* (XOP) specification ([W3C05b]). Using this exclusive element, the attached binary content logically becomes inline with the SOAP document even though actually it is attached separately.

Furthermore, MTOM optimized messages always are valid messages in terms of SwA, since the actual message format is identical. The only difference is the kind of referencing the binary content. This way, compatibility of MTOM messages with SwA endpoints is guaranteed.

After all, the Web service API of Cloud42 was designed to support MTOM optimized messages, since this is the most modern and the most powerful standard and tool support is no problem in general.

In order to provide the caller with potential error messages, the RemoteResult mechanism introduced in Section 3.1.2 was applied, too.

3.2.2 Amazon S3

As mentioned several times throughout this work, the Amazon Web Services do not only consist of EC2, but of several related services, too. One of them is Amazon S3, which allows storing data in the cloud.

Of course, copying data from a location on S3, a so called "bucket", into an AMI instance is interesting for users of EC2, since connections are very fast and large amounts of data can be handled quickly. For instance, S3 can be used to store instance-specific data or configurations.

There are a lot of tools available that help to control S3 and to exchange data between AMI instances and buckets. By installing one of these tools on his AMI and by invoking it using Cloud42's generic remoting capabilities as described in Section 3.1, a user can take advantage of S3 without any limitations. For instance, Listing 3.3 illustrates a request to copy the contents of the whole folder "etc" into the bucket "mybucket" using the prefix "pre".

Listing 3.3 File transfer to S3

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soapenv:Header/>
  <soapenv:Body>
    <web:executeCommand>
      <web:dnsName>ec2-67-202-53-93.compute-1.amazonaws.com</web:dnsName>
      <web:rsaKey>-----BEGIN RSA PRIVATE KEY----- ... </web:rsaKey>
      <web:command>s3sync.rb -r /etc mybucket:pre</web:command>
    </web:executeCommand>
  </soapenv:Body>
</soapenv:Envelope>
```

3.2.3 Passing URLs

In spite of the flexibility given by the general remoting mechanisms, Cloud42 also provides a function that allows copying files from a S3 bucket to an AMI instance directly, without the user having to install separate tools. Therefore, it uses the fact that S3 allows generating unique URLs for objects in S3 buckets. Such an URL can be passed to the file upload function of the Cloud42 API.

This way, there is no restriction on S3, but it is even possible to upload files from any location using HTTP or FTP.

Listing 3.4 provides a sample request for uploading a file from another web server.

Listing 3.4 Uploading a file from an URL

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soapenv:Header/>
  <soapenv:Body>
    <web:uploadFileFromURL>
      <web:dnsName>ec2-75-101-230-49.compute-1.amazonaws.com</web:dnsName>
      <web:rsaKey>-----BEGIN RSA PRIVATE KEY----- ...</web:rsaKey>
      <web:targetDir>/etc</web:targetDir>
      <web:targetFilename>myConfig.xml</web:targetFilename>
      <web:url>
        http://my.files.com/myConfig.xml
      </web:url>
    </web:uploadFileFromURL>
  </soapenv:Body>
</soapenv:Envelope>
```

3.2.4 Providing Files at Startup

Ideally, information such as configuration files or specific instance data can not only be provided when an instance is already running, but right on startup. This way, the information can be used to configure running applications or to load certain data during boot process.

The EC2 API enables the user to upload additional user-data when starting an AMI instance ([LLC08]). This data can either consist of simple String values or whole binary files up to a limited size (currently 16K).

Cloud42 as a wrapper around the EC2 API offers the same feature.

Although the strict limitation of file size disallows passing all required instance data directly in most cases, this mechanism is very powerful. Passed data can be easily read on the instance, thus allowing to be processed in a startup script.

For example, an URL or the name of a bucket on S3 can be specified as user-data at startup. On the AMI instance, a startup script is used to download the file from the given location. This way, generic AMIs can be created that retrieve their actual data or configuration not until startup.

There are two papers discussing the mechanism sketched above. In [Cab08], PJ Cabrera illustrates how S3 can be used to provide data dynamically, for instance in order to always install the latest software packages. In a more general article, he describes a generic way to customize an AMI instance at startup ([Cab07]).

3.3 Bundling new AMIs

Whenever a change on a running AMI instance is made, bundling a new AMI is neccessary in order to keep these changes.

However, bundling an AMI is a long-running process, mainly consisting of two steps, both of which are executed using the Amazon AMI Tools installed on each AMI (see Section 2.3.3). At first, the image of the current AMI is created and stored on the AMI itself. Second, it is uploaded into a bucket on S3. Each of these steps takes some time, depending on the actual size of the AMI to bundle.

Because of this tediousness it is desired to be able to initiate the process of bundling a new AMI remotely, for instance using an asynchronous Web service call. Furthermore, this is a possibility to take advantage of the notification mechanism that will be introduced in Section 3.4. As soon as the bundling was completed, an AMI can send a notification message on a topic previously defined by the caller thus allowing to react on the event. For instance, the newly bundled AMI could be registered and started.

Of course, this could be achieved by executing a sequence of several remote commands using the Cloud42 API. However, for convenience reasons, the Web service API of Cloud42 provides a function encapsulating the whole process. Listing 3.5 shows a sample request.

Listing 3.5 Sample request for bundling a new AMI (not MTOM optimized)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:web="http://webservice.cloud42.jw.de"
    xmlns:xsd="http://domain.core.cloud42.jw.de/xsd">
   <soapenv:Header/>
   <soapenv:Body>
      <web:bundleImage>
         <web:dnsName>ec2-75-101-244-126.compute-1.amazonaws.com</web:dnsName>
         <web:rsaKey>-----BEGIN RSA PRIVATE KEY----- ... </web:rsaKey>
         <web:credentials>
            <xsd:awsAccessKeyId>...</xsd:awsAccessKeyId>
            <xsd:secretAccessKey>...</xsd:secretAccessKey>
            <xsd:userID>...</xsd:userID>
         </web:credentials>
         <web:targetBucket>myCloud42Bucket</web:targetBucket>
         <web:newImageName>img3</web:newImageName>
         <web:is64Bit>false</web:is64Bit>
         <web:notifyWhenFinished>true</web:notifyWhenFinished>
         <web:topic>myTopic</web:topic>
         <web:messageText>Bundling completed.</web:messageText>
         <web:messageInfo>123-acs</web:messageInfo>
         <web:keyFile>cid:1144418762956</web:keyFile>
         <web:certFile>cid:1075376821095</web:certFile>
      </web:bundleImage>
   </soapenv:Body>
</soapenv:Envelope>
```

In order to be able to execute the AMI Tools on the instance, all authorization information must be passed. Furthermore, the user's AWS key and certificate must be sent because it is needed to sign the new AMI. As described in Section 3.2.1, these files can be attached to the request message using MTOM.

By setting the corresponding parameters, the user has full control over the notification that will be sent when bundling has finished. Once again, the RemoteResult concept (see Section 3.1.2) is used here in order to provide a response that can be parsed und understood easily by non-human clients.

3.4 Notification Mechanism

As stated several times throughout this work, a computing cloud is dynamic by its nature. In particular, this means it can aquire new resources on demand. On the other hand, resources that are no longer needed must be released as soon as possible. In order to achieve such dynamics, a lot of communication is necessary. The states of different computing resources must be exchanged frequently and there must be some kind of controller that evaluates these messages and acts accordingly.

Descending one level of abstraction and speaking in terms of EC2, a user running several AMI instances on EC2 may want to react on events on his instances. To give an example, if the load of instance A exceeds a particular value, he may want to start another instance of the same type. Or, to give another example, as soon as the extensive calculations he rolled out to his instance are finished, he wants that instance to be terminated.

This raises the question how such information can be passed to the user.

Polling is the most obvious choice. The user checks the required information at regular intervals and reacts as soon as his check is "successful".
However, this approach has a lot of disadvantages, from which only two shall be conducted here.

The first one is the fact that polling causes a lot of overhead. In particular, if the expected event occurs only rarely or even never, lots of bandwith and other resources are wasted just searching for something that is not there.

Second, the polling interval is critical. If it is too short, the mentioned resource consumption becomes a serious problem. If it is too long, one risks missing the monitored event or at least it could be too late to react on it.

So a push based notification architecture would be much more preferable. It is desirable that (only) interested participants are notified as soon as an event occurs. In order to achieve this, the publish/-subscribe pattern could be used in combination with a topic mechanism, allowing to subscribe to notification messages triggered by certain events.

The above mentioned "user" (or event sink in terms of the publish/subscribe paradigm) must not necessarily be a human. Of course, it could be the system administrator equipped with his mobile phone. But also consider it being a piece of software or a BPEL process automatically orchestrating EC2. As a conclusion, notification messages must be available in arbitrary formats and via arbitrary transportation protocols, so that processing data is possible at each particular event sink.

The goal of this section is to introduce a notification mechanism integrated in Cloud42 that does exactly fulfill all the previously collected requirements.

3.4.1 General Architecture

Figure 3.3 shows the actors and the general architecture of the Cloud42 notification mechanism.

Since one of the requirements was supporting different message formats and different transport protocols when sending notification messages to the sinks, there must be a central point that coordinates

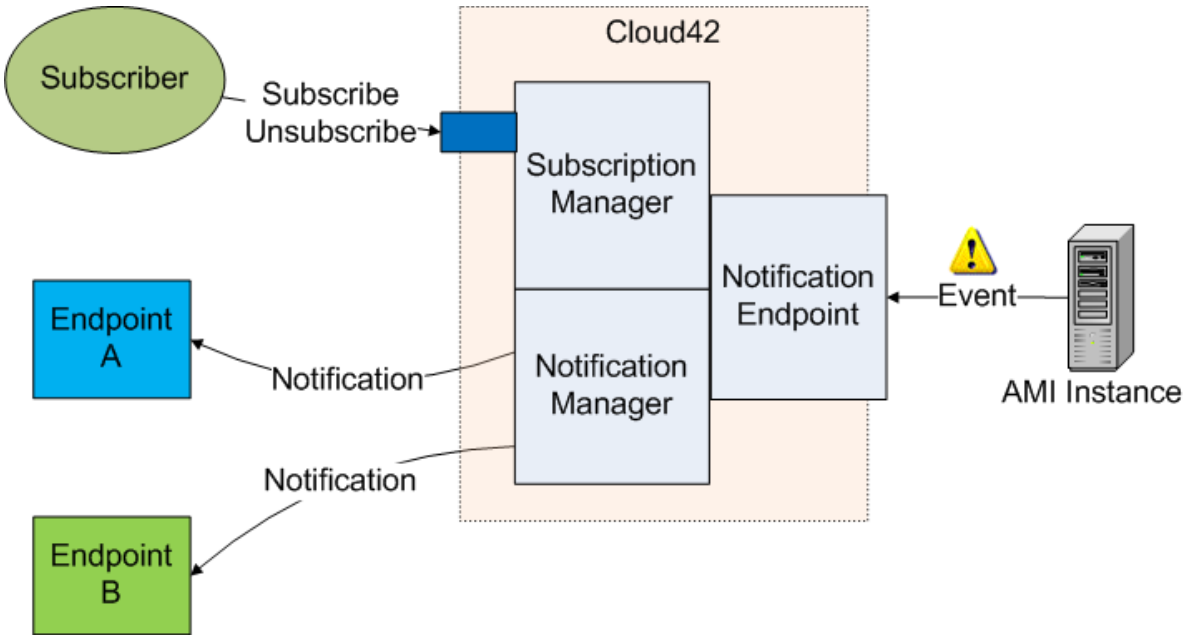

Figure 3.3: Notification mechanism - architecture

subscribing endpoints with their properties and transforms these messages into the desired formats. Therefore, sending messages directly from the AMI instances to the interested endpoints was not an option.

Instead, an approach was chosen featuring a "Notification Core" within Cloud42. This way all application logic is concentrated at one single place and kept away from the EC2 instances, so that they can be run fully independently and without any configuration. The core of the notification mechanism consists of three parts that can be separated logically.

The Cloud42 Web service interface provides a method that allows a subscriber to register for notification messages on a certain topic. In this subscription request, an arbitrary endpoint kann be passed as data sink that finally receives the notifications.

The subscription request is processed by the SubscriptionManager part of the core, which is responsible for handling subscription and unsubscription requests as well as for storing the subscribed endpoint references.

Second, an endpoint is offered that receives and processes event messages coming from an EC2 server instance. The address of this endpoint can be configured using the Cloud42 Web service interface. Event messages from AMI instances always have to be sent to the endpoint address specified this way.

Whenever a message arrives at this endpoint within Cloud42, the third part, the NotificationManager is invoked. Its job is to go through the list of subscribed endpoints for the message's topic and to package the message data into the desired format for each particular endpoint. Furthermore, it finally sends the notification messages to the sinks using the respective transportation protocol.

3.4.2 Alternative Approaches

There are two standards that are strongly related to the work at hand. Each of them deals with the notification issue and related paradigms such as publish/subscribe in a slightly different manner.

WS-Notification

The first one to mention is WS-Notification (see [OAS]). WS-Notification is a "family of related white papers and specifications that define a standard Web services approach to notification using a topic-based publish/subscribe pattern" ([GNC+]). Of course, the standard also covers quite simple scenarios like the Cloud42 notification mechanism so that using a WS-Notification based approach would be a reasonable alternative.

However, using WS-Notification was not the method of choice, mainly due to the following facts:

- WS-Notification is a very wide standard that is embedded into the WS Resources Framework, a set of proposed standards that formalizes the relationship between Web services and state. It is quite comprehensive and complex. Because of this, it would be oversized for a simple scenario like this one where only a simple publish/subscribe solution is necessary. Advanced notification paradigms are far beyond the scope of the requirements forming the base of this work.
- The elaborated EC2 management framework should be put into practice by actually implementing it. Unfortunately, tool support for WS-Notification is rarely available. So it would be inevitable to implement at least parts of the standard, which is a very time consuming undertaking.

WS-Eventing

Another standard that enables interoperable publish/subscribe systems is WS-Eventing (see [W3C06]). Contrary to WS-Notification, this specification is rather short and lightweight. It defines "a protocol for one Web service (called a "subscriber") to register interest (called a "subscription") with another Web service (called an "event source") in receiving messages about events (called "notifications" or "event messages")" ([W3C06]). By introducing a so-called "subscription manager", the subscription can be renewed or canceled by the subscriber. By default, the subscription ends after a certain amount of time.

At a first glance, this standard seems to be perfectly suited to be used in the Cloud42 notification mechanism. Indeed, the application of WS-Eventing would make sense, since the standard defines all the actions and message flows that are needed (and some more, of course). Also, it supports an abstraction from concrete transportation protocols and message formats by identifying endpoint references using WS-Adressing and by supporting different delivery modes.

But again, there is one drawback: the lack of a working implementation. Although there is a module for the Web service engine Axis2 called "Savan"[1] that claims to implement the complete WS-Eventing standard, several tests stated it as buggy, undocumented and only rudimental working. So it would be

[1] `http://wso2.org/projects/savan/java`

necessary to develop a new WS-Eventing implementation from scratch, a task that surely exceeds the scope of this work.

Taking everything into account, an architecture containing some elements from WS-Eventing and some elements from WS-Adressing for handling endpoint references was chosen to fit best.

3.4.3 The Notification Core

Figure 3.4 provides a deeper look into the Notification Core and its three components SubscriptionManager, NotificationManager and Endpoint.

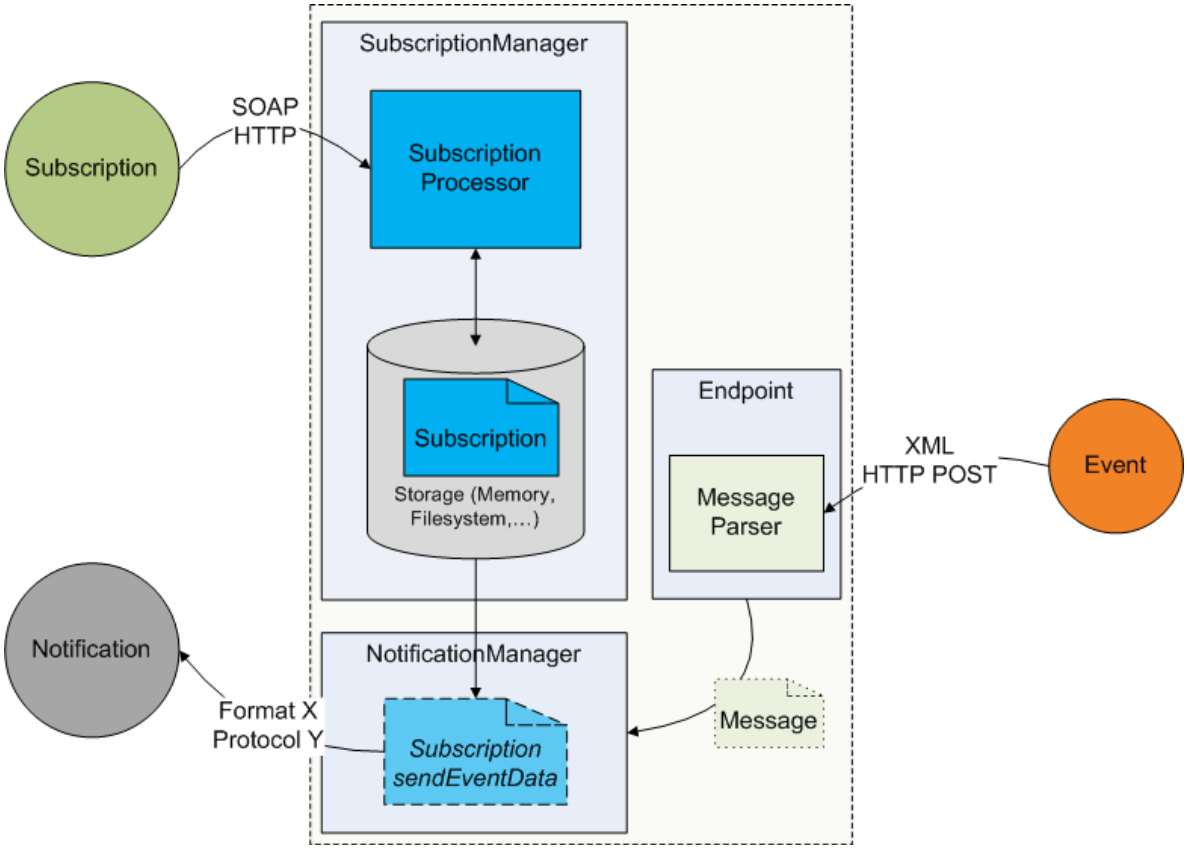

Figure 3.4: Notification Core

Subscribe and Unsubscribe

Subscription requests as well as unsubscription requests are SOAP messages that are processed by a *SubscriptionProcessor*. The SubscriptionProcessor is responsible for parsing the SOAP message and gathering the required information, depending on the kind of subscribing endpoint. A subscription message itself consists of two parts, a fixed part telling the system which SubscriptionProcessor to use

and to which topic to subscribe. The variable part contains an endpoint reference in a format that is understandable for the respective SubscriptionProcessor.

Listing 3.6 shows an example for such a subscription message.

Listing 3.6 Example for a subscription request message

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soap:Header/>
  <soap:Body>
    <web:subscribe>
      <web:subscriptionProcessor>
         de.jw.cloud42.core.eventing.subscriptionProcessor.SOAPSubscriptionProcessor
      </web:subscriptionProcessor>
      <web:topic>myTopic</web:topic>
      <!-- web:subscriptionMessage contains the variable part -->
      <web:subscriptionMessage>
        <wse:Subscribe xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing">
          <wse:Delivery>
            <wse:NotifyTo>
              <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
                http://localhost:8085/monitor
              </wsa:Address>
            </wse:NotifyTo>
          </wse:Delivery>
        </wse:Subscribe>
      </web:subscriptionMessage>
    </web:subscribe>
  </soap:Body>
</soap:Envelope>
```

The first two child elements of the `<web:subscribe>` element contain the information which actual SubscriptionProcessor should handle the request and to which topic the subscription belongs.

The second part, provided within `<web:subscriptionMessage>`, is the specific part that is parsed by the SubscriptionProcessor. In this case, it consists of a subscription request according to the WS-Eventing specification ([W3C06]). The `SOAPSubscriptionProcessor` used in this example is intended to be used for subscribing endpoints that receive their notifications as SOAP messages over HTTP.

Appendix A demonstrates the extensibility of the concept and shows another SubscriptionProcessor that can handle notifications via SMS.

Having parsed the subscription message, the SubscriptionProcessor creates a *Subscription* object accordingly (in our example, it would be a `SOAPSubscription`) and stores it in the subscriber storage, linked to its topic. The subscriber store may be implemented in different ways, however in the current implementation it uses a database to persist the Subscriptions.

The Subscription entity holds information such as the endpoint reference for notification messages. Furthermore, it contains the logic to transform messages from the internal message format into the desired format and to send it to its endpoint. Getting back to our example, it would wrap the message

into a SOAP envelope and send it to the designated endpoint `http://localhost:8085/monitor` using HTTP.

As a response to the subscription request, the subscriber retrieves a SOAP message containing a subscription id that must be used to identify the subscription when unsubscribing it. This subscription id is unique for each endpoint on each topic, meaning that the same endpoint can subscribe to various topis, but always is identified by another id. Listing 3.7 illustrates an example response to a subscription request.

Listing 3.7 Response to subscription request

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <ns:subscribeResponse xmlns:ns="http://webservice.cloud42.jw.de">
      <ns:return>uuid:99733DEDFB3443D5281222782013924</ns:return>
    </ns:subscribeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.8 shows the according unsubscribe request.

Listing 3.8 Example for a unsubscribe message

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soap:Header/>
  <soap:Body>
    <web:unsubscribe>
      <web:subscriptionId>uuid:99733DEDFB3443D5281222782013924</web:subscriptionId>
    </web:unsubscribe>
  </soap:Body>
</soap:Envelope>
```

Since ending a subscription requires nothing more than deleting an entry from the list of subscriptions, there is no need to specify a special SubscriptionProcessor.

For the sake of completeness, the response to the unsubscribe request is presented in Listing 3.9.

Listing 3.9 Response to unsubscription request

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <ns:unsubscribeResponse xmlns:ns="http://webservice.cloud42.jw.de">
      <ns:return>true</ns:return>
    </ns:unsubscribeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Event Messages

The last aspect to point out is the communication between the AMI instance and the Notification Core.

Event messages from the EC2 server instance must be sent to an endpoint offered by Cloud42 in a predefined XML format using a HTTP POST. The Cloud42 Web service interface offers methods to configure this endpoint.

Thanks to using the HTTP POST method, the payload can be transferred in the body of the event message (compared e.g. to a HTTP GET where it would be necessary to put it in the header/the request URI) allowing to send larger amounts of data. Furthermore, a POST message resembles the principle of a RESTful Web service according to the REST architecture ([Fie00]).

For instance, shell scripts can be used on the instance to easily trigger such messages. The command line tool *Curl*, available for all Linux based systems, can be invoked very easily. [LLC08] illustrates how instance metadata such as the instance id can be accessed for usage within the messages. This way, absolutely no configuration on the EC2 server instance is necessary. The only thing that must be known is the address of the Cloud42 endpoint where to send the event messages.

Listing 3.10 illustrates a HTTP POST message containing some event data.

Listing 3.10 HTTP POST event message

```
POST /messages HTTP/1.1
User-Agent: curl/7.16.3
Host: 127.0.0.1:8088
Accept: */*
Content-Type: text/xml
Content-Length: 151

<message xmlns="http://cloud42.jw.de/message">
  <topic>myTopic</topic>
  <instanceId>i-38e24051</instanceId>
  <timestamp>2008-09-30 11:19:10 GMT</timestamp>
  <text>message text</text>
  <info>additional message info</info>
</message>
```

The first thing to mention is that messages are transferred using the Content-Type `text/xml` to identify their payload as XML data.

The message itself simply consists of a `<message>` element having five child elements carrying the actual event information, including the topic. The topic field is required, all the other fields can be used arbitrary depending on a user's need. For instance, the `<info>` element could contain a code or an identifier that is picked up by the final receiver of the event message (a subscribed endpoint) for further processing. All values are simple Strings for full flexibility.

The XML schema is presented in Listing 3.11.

As soon as such an event message arrives at the Endpoint component of the Notification Core, it is parsed and transformed into an internal representation. Now the NotificationManager is called, passing this message. Its job is to retrieve all Subscriptions for the current topic from the subscription storage and to initiate the process of sending the notification messages. As explained above, each particular Subscription knows how to transform the event message from the internal format into a format understandable by its endpoint and how to send it there.

Listing 3.11 XML schema for event messages

```
<?xml version="1.0" standalone="yes"?>
<xs:schema targetNamespace="http://cloud42.jw.de/message"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="topic" type="xs:string" minOccurs="1"/>
        <xs:element name="instanceId" type="xs:string" minOccurs="0"/>
        <xs:element name="timestamp" type="xs:string" minOccurs="0"/>
        <xs:element name="text" type="xs:string" minOccurs="0"/>
        <xs:element name="info" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The whole workflow process, both for processing a subscription request and handling an event message, is summarized again in the sequence diagram in Figure 3.5.

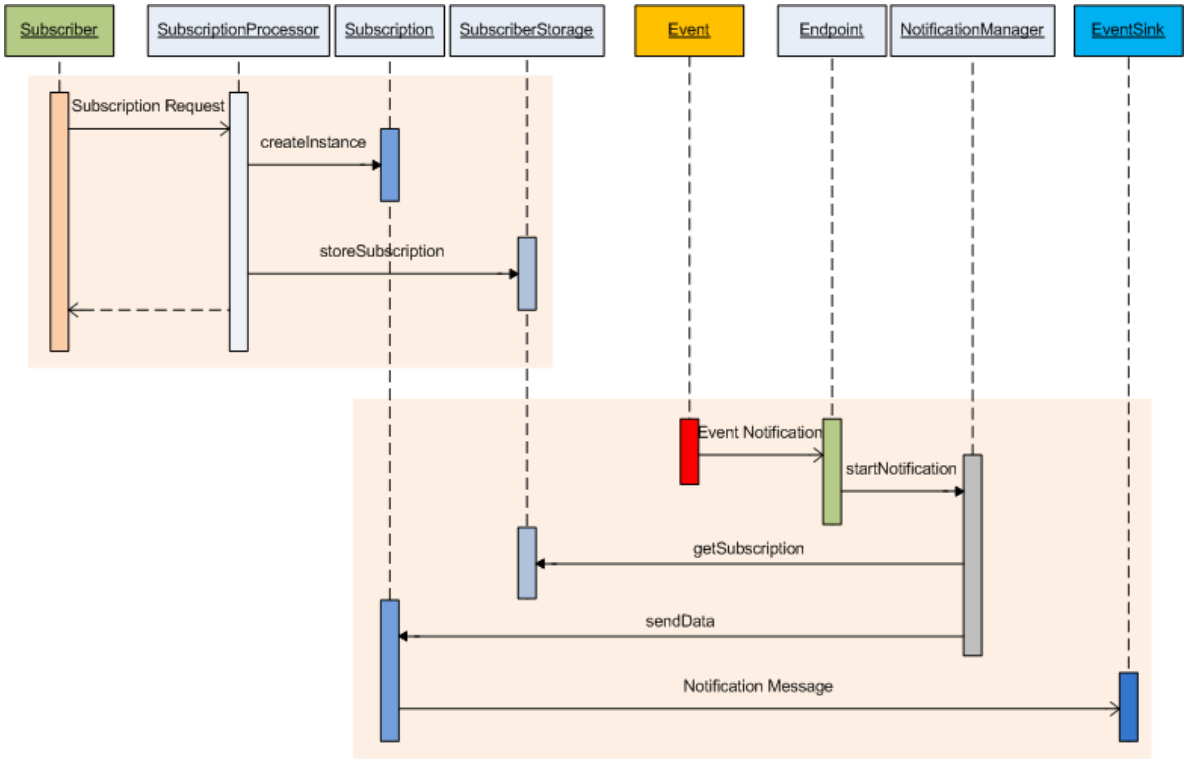

Figure 3.5: Sequence diagram: subscription and event processing in Notification Core

3.4.4 Extensibility

The concept of the SubscriptionProcessor and the specific Subscription objects was elaborated in order to empower protocol specific implementations of the algorithms used to transform and send notification messages. Due to the fact that all protocol specific processings are concentrated at one place, it is very easy to hook in user-defined SubscriptionProcessors for arbitrary message formats. By specifying the desired SubscriptionProcessor in the subscription request on a per subscription basis, it is possible to have different kinds of endpoints subscribed at the same time.

This way, extensibility is guaranteed and the Cloud42 notification mechanism becomes adjustable for every purpose. For instance, Appendix A illustrates how to implement a SubscriptionProcessor supporting SMS notification with few lines of code.

3.4.5 Message Filtering

One aspect of a publish/subscribe system that was not covered yet is filtering. In some cases, not all subscribers are interested in all messages on a particular topic they subscribed to. For instance, a subscriber may only want to retrieve event notifications from a specific instance.

Although it is not realized in the actual implementation, the concept elaborated in this section is powerful and flexible enough to be easily enhanced to provide filtering capabilities. There are two ways to achieve this goal.

In order to pass just some selected messages to a subscribed endpoint, it is possible to implement user-defined SubscriptionProcessors and Subscription classes that do some kind of filtering before transforming and sending the event messages.

In most scenarios, this is an acceptable approach, but there is one disadvantage: this way, the filter becomes protocol dependent. Filtering and message processing are coupled together making it necessary to implement a new SubscriptionProcessor for each type of filter and vice versa.

Instead, as a second approach, the idea of specifying the SubscriptionProcessor only in the subscription request message could be applied for specifying a filter class in this message, too. Listing 3.12 demonstrates such a fictive request message.

In this solution, filtering logic is seperated from subscription processing. The filter class is attached to the Subscription object offering the possibility to assign different filters to different subscriptions. A filter is applied by invoking a predefined method on its filter class before the notification message is transformed and sent.

Listing 3.12 Subscription request message with filter class

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soap:Header/>
  <soap:Body>
    <web:subscribe>
      <web:subscriptionProcessor>
        my.company.subscription.MySubscriptionProcessor
      </web:subscriptionProcessor>
      <web:topic>myTopic</web:topic>
      <web:filter>my.company.filters.MessageFilter</web:filter>
      <web:subscriptionMessage>
        <!-- specific content for MySubscriptionProcessor -->
        ...
      </web:subscriptionMessage>
    </web:subscribe>
  </soap:Body>
</soap:Envelope>
```

Chapter 4

Implementation

In this chapter, the implementation of Cloud42 is described.

At first, the requirements are summarized shortly. After that, the general architecture of Cloud42 is illustrated and the work at hand provides a look into the technologies that were chosen for implementing the management tool, discussing their pros and cons. Finally, some of the most outstanding features concerning the user interface and the Web service interface are presented. The section describing the Web service interface contains a practical example how Cloud42 can be of use in a real world scenario by demonstrating how to embed its functionality into a BPEL workflow.

4.1 Requirements

This section handles the requirements that formed the basis for developing Cloud42. As already outlined in the Motivation (see Section 1.1), there is more than one reason justifying a new, comprehensive management tool for EC2.

4.1.1 Functional Requirements

Functionalities

Of course, a management tool must provide some serious functionalities in order to be able to accomplish certain tasks in a convenient way. In the case of EC2, this mainly means that AMI instances must be started, monitored and terminated. Therefore, the following functionalities build the core of Cloud42 and are summarized as the "base functions":

- List existing AMIs
- Register new AMIs (by specifying their location on S3)
- Deregister existing AMIs
- Create, edit, list and delete RSA keypairs for securing access to AMI instances

- Create, edit, list and delete security groups used to control rights on running instances
- Run instances supporting all arguments understood by the EC2 API
- List running instances with their data
- Reboot running instances
- Terminate running instances

Please note that when this work was planned and started in spring 2008, some of Amazon's newest features such as Elastic IPs or the Amazon Elastic Block Store were not yet available, so these features seem to be missing. Indeed, they neither were part of the requirements nor are implemented in this first version. However, it is not a problem to enhance Cloud42, so they are going to be implemented soon in an upcoming second version.

Apart from the listed base functions, four extended functionalities were considered to be useful. These are:

- File transfer functionalities allowing to copy arbitrary files on AMI instances
- Support for automated bundling of new AMIs
- Controlling instances executing shell commands and uploading batch files remotely
- Support for a notification mechanism allowing to subscribe endpoints to receive event messages from running instances

These requirements are not as accurate as the base functions and would require further investigation, but since they are extensively discussed in Chapter 3, no further description is given here.

Actors

There are two different kind of actors that must be able to interact with Cloud42.

At first, an user interface must be provided in order to allow an human user to use the system.

Second, Cloud42 must provide a Web service interface. One of the main motivations behind this work is the idea to orchestrate EC2 using BPEL workflows. Not only for this purpose, it is inevitable that all management functions are accessible as Web service with a clean and comfortable design.

4.1.2 Non-functional Requirements

There are no specified response times or other typical non-functional requirements that Cloud42 must fulfill.

4.1.3 Technical Requirements

In order to be accessible from multiple locations and to support various devices (such as PDAs), the user interface must be web-based, allowing system administrators to manage their EC2 instances from everywhere. Furthermore, being a web application, Cloud42 itself can be run on the cloud which is an interesting idea because it allows controlling the cloud from within the cloud.

As programming language Java has to be used, ideally in combination with some state-of-the-art technologies like outlined in the following section. Java was selected because it has proven to be an ideal choice and an absolute standard for implementing large-scaled web applications.

4.2 Architecture

The following section describes the architecture of Cloud42. It does not cover each detail, but it illustrates the main goals and the main concepts used when designing Cloud42.

4.2.1 Goals

There are some goals that the architecture of Cloud42 aims for. Some of them are typical for an Open Source project, some of them were inspired by the nature of the project itself. Mainly, these design goals are:

- **Extensibility**
  Intended to be published under an Open Source licence, Cloud42 should provide some extension points where other developers can trim or enhance it to fulfill their individual needs. Ideally, such extensions or adjustments can take place without having to touch any existing code. A good example for the extensibility of Cloud42 is the notification mechanism described in section 3.4 and appendix A.
- **Modularity**
  Since Cloud42 has two different kinds of interfaces - a graphical user interface and a Web service interface - that share the same business logic and domain objects, modularity was an important architecture goal. The web application providing the user interface and the Web service layer should be separated from each other in order to reach full flexibility. However, both layers should be able to access the same implementation of the business logic and the domain objects. Furthermore, a fine-granular modularity allows to split the application into different, logically separated modules even within one application layer.
- **Ease of Use**
  In order to become a widely accepted tool, Cloud42 must be easy to handle. This topic can be partitioned into two aspects. At first, it must be easy to install and deploy Cloud42 without having to set up complicated environments or database servers and without having to download additional tools. Ideally, Cloud42 should be able to run "out of the box". Secondly, both the graphical user interface and the Web service interface should be easy to understand and comfortable to use. For

the user interface this means the presence of filtering and sorting functionalities and a common look&feel in general. The Web service interface should provide clearly named methods with well named parameters and return values.

4.2.2 General Architecture

Figure 4.1 shows a high-level deployment diagram of Cloud42.

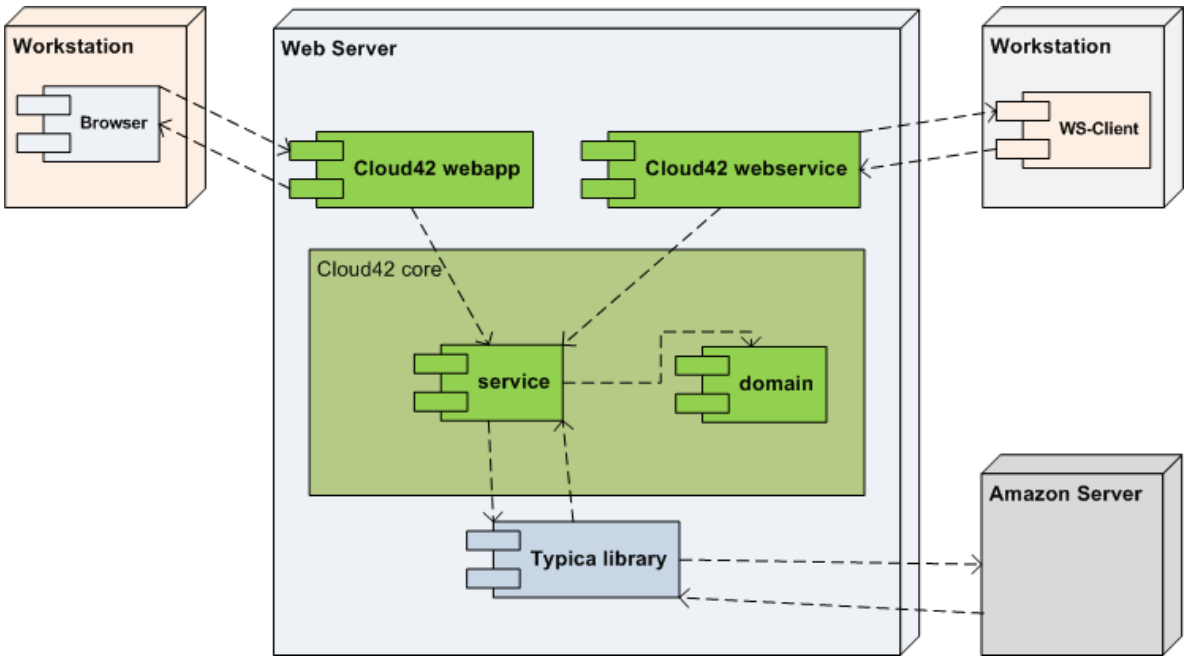

Figure 4.1: Deployment diagram of Cloud42

Basically, Cloud42 is split into the three modules `Cloud42 webapp`, `Cloud42 webservice` and `Cloud42 core`.

The `webapp` module contains the Cloud42 web application, implemented using the MVC framework Java Server Faces and some related technologies (see Section 4.3.1).

In order to save user accounts and AWS credentials, the web application needs a database. By using Hibernate and its abstraction mechanisms, the actual database system can be switched quickly. For instance, an embedded database can be chosen, so that running Cloud42 does not require any real database server, a fact that is essential for the design goal "ease of use".

As its name says, the module `webservice` represents the implementation of the Web service layer by using the Web service engine Axis2 that is described in Section 4.3.5. Mainly, this module wraps the functions of the `service` module and exposes them as Web service functions.

`Webapp` and `webservice` are totally separated from each other and it is possible to deploy and use only one of them or both at the same time. This way, the user can adjust Cloud42 to his needs without having to deploy unnecessary modules.

Finally, the `Cloud42 core` is divided into some submodules, of which the most important are `service` and `domain`.

The `service` module contains the real core of Cloud42 - its application logic. In order to access the EC2 API from Amazon, the Java library Typica, a wrapper around this API, is used. A review of Typica can be found in Section 2.3.4. Furthermore, the `service` module references a module called `domain` that simply holds the domain objects for the whole application.

Apart from the two modules described yet, `Cloud42 core` conists of some more modules, each of them containing the implementation of one of the enhanced concepts elaborated in Section 3. This way, the design goal "modularity" is reached.

4.2.3 Design Patterns

Although the architecture of Cloud42 is not pattern-driven like proposed by Buschmann et al. in [BMR+96], some common design patterns are applied.

The most central pattern followed in the design of Cloud42 is the "Don't Repeat Yourself" pattern, also known as the *DRY* pattern. This pattern advices to avoid duplications both in code and configuration files. At least concerning duplications in code, the concept of modularity helps very well to realize this pattern. Whenever it would be necessary to repeat a piece of code, it is encapsulated into a new module. Furthermore, Cloud42 provides one global configuration file, so there is no need to adjust settings at different or even multiple locations.

Another design pattern used is the *Factory* pattern. In order to provide extensibility and flexibility, the key idea behind this pattern is to stay as unconcrete as possible when instantiating new objects. In order to achieve this, a so-called Factory method is used that helps hiding the type of an object when it is created. [GHJV95] provides a detailed description of this pattern.

In the case of Cloud42, the Factory pattern is applied in the notification mechanism in order to be able to dynamically instantiate different kinds of SubscriptionProcessor classes. For more details, see Section 3.4.

4.3 Technology Stack

In these days, a developer is spoilt for choice which technology to trust and which framework to choose. There are Open Source frameworks for quite every purpose. The following sections provide a short introduction into the technologies used for implementation as well as the reasons why they were supposed to fit best in the case of Cloud42.

All frameworks are compatible with Java SE 6 that was used as basis. An arbitrary Java compliant application server can be taken to execute Cloud42.

4.3.1 Java Server Faces

Java Server Faces[1] (JSF) is a server side user interface component framework for Java web applications. Actually, it is just a specification, but there is an official reference implementation, too. Released already in May 2004, it is quite an old framework. The recent version 1.2 is dated to May 2006 and enjoys a great popularity even in these days, being on its way to be a de-facto standard.

Based on the proven Servlet technology, Java Server Faces realizes the MVC pattern and its architecture is based on some other popular design patterns (see [Jos05]). Figure 4.2, taken from the online documentation, shows the relation of Java Server Faces to Java Server Pages (JSP).

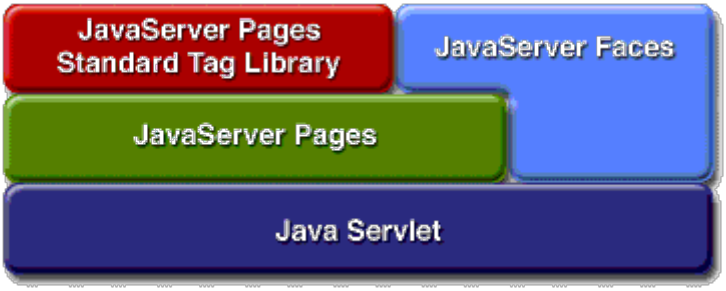

Figure 4.2: Java Server Faces in relation to Java Server Pages

The main benefit of Java Server Faces compared to Java Server Pages is its cleaner architecture and its more comprehensive approach. When using Java Server Faces there is no more need for complementing MVC frameworks such as Struts[2].

One of the most important concepts behind Java Server Faces are the *UI components*. The user interface is composed of various components, such as textboxes, buttons or dropdown lists. Components have properties like a caption, different background colors etc.. Whenever a component changes its state, e.g. when a user clicks a button, an event is fired that can be used to invoke some server-side Java code.

There is a large set of component libraries available, some of them even featuring build-in AJAX support. Speaking in terms of the MVC pattern, the components represent the View.

The Model is composed of so-called *Backing Beans*, simple POJOs (Plain Old Java Objects, means simple Java classes) whose properties are bound to properties of the components. Furthermore, events from the UI can be bound to methods of the Backing Beans.

Finally, the *FacesServlet* is the Controller that defines application behavior by mapping user actions to model updates and by selecting views for response.

A request is processed within the JSF lifecycle, split in six phases as illustrated in Figure 4.3.

The central element is the *component tree* which represents the structure of the (possibly nested) controls of the submitted view. In the *Restore view* phase, this component tree is build, either from

[1] `http://java.sun.com/javaee/javaserverfaces/`
[2] `http://struts.apache.org/`

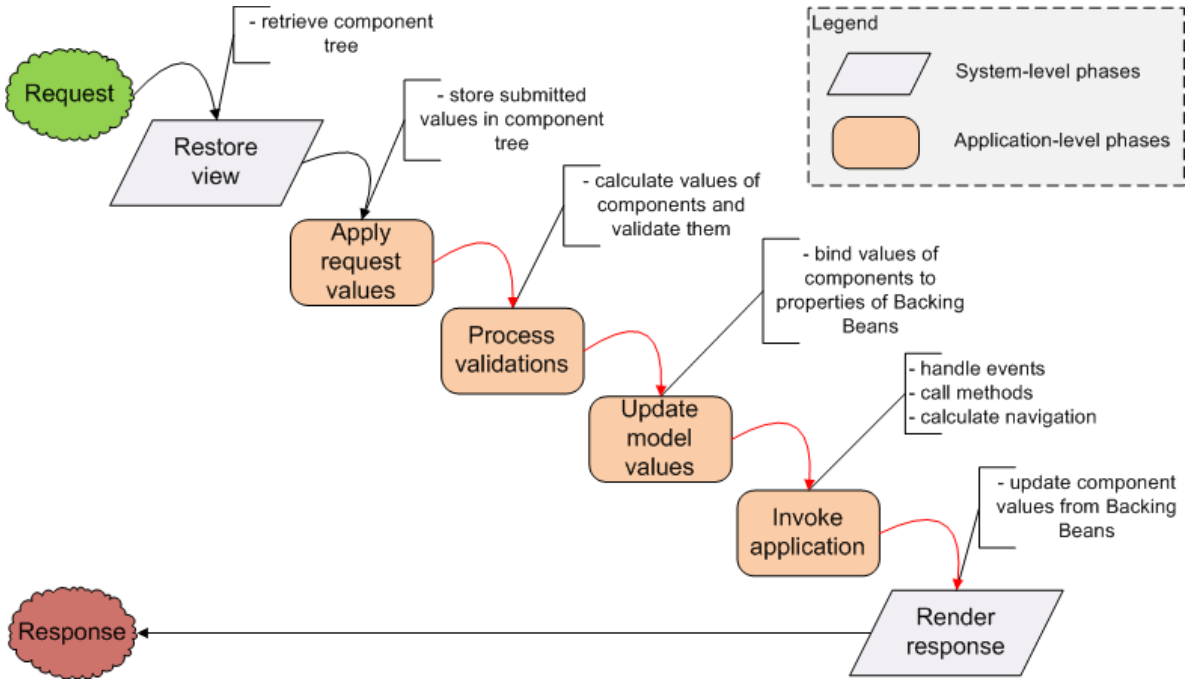

Figure 4.3: The JSF lifecycle

the view definition if it is requested the first time, or from a previous request if it already contains some user data. The next steps modify the component tree according the the values submitted with the request. Finally, the properties of the Backing Beans are set and the application logic is invoked. As a last step, the (now filled) component tree is rendered as HTML and sent back to the requestor, usually the browser.

Understanding the JSF lifecycle is essential when developing Java Server Faces applications since not each property holds correct values in each phase. For instance, the values of the Backing Beans should not be accessed until the *Invoke application* phase, because they are updated in the preceding *Update model values* phase. It is possible to skip some phases by instructing the FacesServlet accordingly.

Being a state-of-the-art web framework and providing some advantages over JSP, especially in combination with the technologies described below, JSF was chosen for implementing Cloud42.

4.3.2 Facelets

Facelets is an alternative view technology that allows to use JSF without JSP.

Traditionally, one would use Java Server Pages tags for defining the view of a Java Server Faces application, but since JSP has some limitations in the context of JSF, Facelets is the more modern, recommended technology.

Usually, Java Server Pages technology is used to mix static and dynamic web content, pulled from resources made available by other parts of the application; for instance a servlet. Its elements are

processed in a page in one pass from top to bottom with the basic objective of creating a response to a request.

However, Java Server Faces has a more complex lifecycle (see Figure 4.3), and component generation and rendering happen in clearly separated phases. But when JSF is used with JSP, the component creation and rendering are executed in parallel, a fact that can cause trouble in some scenarios: content may appear out of order or not at all. Handling such issues requires a deep knowledge of the individual phases and processing orders, and not each developer may have this knowledge.

When using Facelets, the pages are XHTML documents. Facelets supports templating and the creation of so-called *Facelets taglibs* which represent a reusable set of components.

Listing 4.1 shows a part from a typical view definition.

Listing 4.1 Sample Facelets code fragment (taken from Cloud42)

```
<h:form id="keypairForm">
       <h:panelGrid id="grid" columns="2">
                      <h:outputText value="#{messages.main_keypairPanelKeypairName}" />
                      <h:inputText id="kname" value="#{keypairname}" />
       </h:panelGrid>
       <br/>
       <a4j:commandButton
                      value="#{messages.main_keypairPanelCreate}"
                      action="#{baseFunctionsManager.createKeypair(keypairname)}"
                      reRender="keypairList,material">

              <rich:componentControl for="keypairPanel" operation="hide"
                             event="onclick" />

              <rich:componentControl for="keypairMaterialPanel" operation="show"
                             event="onclick" />
       </a4j:commandButton>
</h:form>
```

An introduction of how to use Facelets and its advantages can be found in [AW08].

Facelets is (unlike JSP, see above) fully integrated into the JSF lifecycle. It uses one of the default extension points: the `ViewHandler`, which is replaced by a `FaceletsViewHandler` that builds views from XHTML documents.

This `FaceletsViewHandler` is invoked in only two of the JSF phases: in *Restore view* and *Render response*. So Facelets touches the first and the last phase of the lifecycle. In *Restore view*, the component tree is build from the XHTML definition of the view. Further processing works as usual, until the *Render response* phase, where the component tree is parsed according to the view definition again and transformed into HTML.

After having evaluated the benefits of Facelets over Java Server Pages, there is no doubt that Facelets is a good choice for the implementation of Cloud42.

4.3.3 JBoss Seam

According to its website[3], Seam is "a powerful new application framework for building next generation Web 2.0 applications by unifying and integrating technologies such as Asynchronous JavaScript and XML (AJAX), Java Server Faces (JSF), Enterprise Java Beans (EJB3), Java Portlets and Business Process Management (BPM)".

As its name may suggest, the goal of Seam is to seamlessly integrate different technologies in order to empower the development of large Enterprise Applications. Inspired by the idea to eliminate complexity at all application levels and to support clean architectures, this integration framework provides some features that are very helpful when developing web applications using Java Server Faces and Facelets.

For instance, Seam tries to solve one of the most important problems traditionally associated with web applications: the handling of state. Seam provides hierarchical, stateful contexts for storing information, ranging from a simple request-based context to long running conversations and business process models using BPM flows. This way, managing state in web applications becomes very comfortable. Furthermore, Seam provides some mechanisms that help to identify and handle clicks on the browser's "Back" button, reducing the risk of submitting critical data twice.

Invented by the developers of the ORM framework Hibernate[4], Seam also provides a deep integration of this framework by unifying session management throughout the application layers. This means, the Hibernate session object can be directly coupled to a session object or a stateful context in the view layer of the application, a fact that significantly reduces problems with Hibernate's Lazy Loading capabilities. In addition, it is possible to use Hibernate Validations for validating user input consistently throughout the whole application (and not only at the ORM layer).

The core of Seam is its annotation mechanism. Simple Java objects are annotated and become *Seam components*, replacing the Backing Beans of Java Server Faces. Using the Dependency Injection pattern, components can be injected into each other. A component lives and holds its data in its predefined state context as mentioned above.

Annotations in general are used heavily when configuring Seam, since Seam's intention is to replace XML configuration files by annotations. The main benefit of using annotations is that they are more readable and that they have to be specified exactly where they are actually taking effect.

Seam offers a lot of other features, including the support for building RESTful applications, integration testing, a Spring integration and many more that are not discussed here. Not all of these features are needed for implementing Cloud42. However, the power of Seam becomes clear even in a relatively small application, mainly by facilitating configuration and state management. Because of this, the Seam framework was used in this work.

[3] `http://www.jboss.com/products/seam`
[4] `http://www.hibernate.org/`

4.3.4 JBoss RichFaces

RichFaces[5] is one of the component libraries for Java Server Faces.

This library contains some ready-to-use components that exceed the standard components of JSF. For instance, RichFaces provides extended tables, lists or dropdown menus. All controls are configurable regarding their look&feel and can be used in conjunction with or without Seam.

But the main feature of JBoss RichFaces is its AJAX integration. By using the Ajax4JSF[6] framework, the component set provides support for enterprise-ready AJAX capabilities out of the box. Some components are AJAX-enabled without the user having to write any line of JavaScript code.

Apart from its own set of components, RichFaces contains some tags that add AJAX features to existing pages and therefore even to standard JSF controls. For instance, it is possible to mark a region as AJAX-rendered (so that this region is automatically refreshed on each AJAX request) or to cause a button to send an AJAX request instead of a normal HTTP POST.

Figure 4.4 illustrates how AJAX capabilities are integrated into the request processing and the JSF lifecycle.

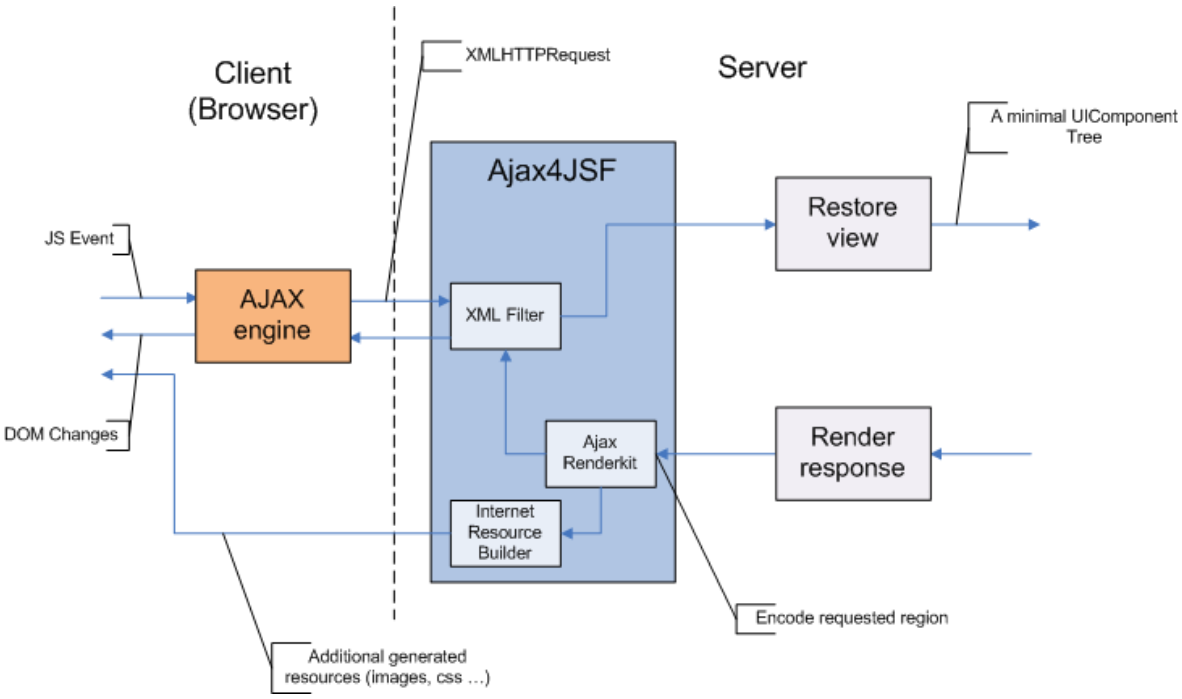

Figure 4.4: Request processing using Ajax4JSF

The Ajax4JSF engine acts as a listener before the first and after the last phase of the JSF lifecycle. Whenever a *XMLHttpRequest*, fired by a JavaScript event handler, is received, the XML message

[5] `http://www.jboss.org/jbossrichfaces`
[6] `https://ajax4jsf.dev.java.net/`

is filtered and a minimal component tree is built, containing only the components that are affected by this AJAX request. This component tree is passed through the JSF lifecycle as usual. Finally, after the last phase, the Ajax Renderkit encodes the changes from the component tree back to a XML representation and the response is sent back to the browser, where the DOM tree of the page is refreshed accordingly.

Unfortunately, this approach has one disadvantage: the whole JSF lifecycle is traversed for each single AJAX request. Although only a minimal component tree is used in order to avoid unneccessary calculations, this can be a performance issue in situations where many incoming AJAX requests need to be processed. So using Ajax4JSF could be critical in scenarios where high scalability is essential.

There are some other component frameworks beside the JBoss RichFaces library. As an example, Apache Tomahawk[7] is worth being mentioned, providing some special components like a file upload dialog. However, none of these alternative component libraries is as comprehensive and as powerful regarding AJAX functionalities as RichFaces. Because of this, and because of the fact that AJAX functionalities are hardly omissible when developing modern, rich user interfaces, RichFaces was selected as component framework for Cloud42.

4.3.5 Axis2

Up to now, technologies for building web applications and user interfaces were covered. Of course, there are frameworks for developing Web services, too.

One of them is the Web service engine Axis2[8]. Apache Axis2 is an implementation of the W3C SOAP submission ([W3C07]), available in C and in Java.

It does not only support SOAP for message exchange, but it also provides capabilities for RESTful Web services. This way, one business logic implementation can offer both a SOAP- and a REST-style interface at the same time.

Axis2 is a re-write of an older engine called Axis. Elaborated in August 2004 and inspired by the experiences made with Axis, the architecture of Axis2 provides many improvements, especially regarding configurability and flexibility. [Fou08] gives an overview of the architecture and the concepts behind Axis2.

One of the core concepts is modularity. Axis2 is designed to support extensibility by plugging in so-called "Modules" that enhance its functionality. For instance, there are currently modules available supporting

- WS-ReliableMessaging
- WS-Coordination and WS-AtomicTransaction
- WS-Security
- WS-Addressing.

[7] `http://myfaces.apache.org/tomahawk/index.html`
[8] `http://ws.apache.org/axis2/`

There is even a module for WS-Eventing that was tested in this work (see Section 3.4.2).

The presence of such modules is the main reason why Axis2 was preferred over alternatives such as the JAX-WS API. By being flexible and extensible, no limitations for further developments are given.

4.3.6 The Maven Build System

In order to manage the build process of a software system consisting of multiple modules and using different packaging mechanisms (for instance, .war files for deployment on an application server and .jar files containing the business objects), a powerful build system is needed. The most popular one in the world of Java is Apache Ant[9].

However, there is a an alternative enjoying steadily growing popularity: Maven[10]. Actually, Maven is not only a build system, but a comprehensive software project management tool. It can not only manage the build process. Even if it is not its primary objective, Maven also can be used to create project reports and complete documentations on a comfortable way.

The main idea behind Maven is its project object model (POM). All project related information is noted in XML files at a central point. If a project is organized in different modules, each module has an own description file and it is possible to define two kinds of relations between these modules:

- a parent-child relation meaning that a module is a sub-module of another module and inherits all settings from its parent
- a simple dependency, meaning that a module depends on another module. Technically, the referenced module is put into the classpath, but no project settings are touched

This way, a tree representing the project structure is built and Maven automatically organizes the build order of the different modules.

Of course, it is not only possible to reference modules belonging to an own project, but also third-party modules and libraries. In order to realize this global dependency management, the concept of *Maven Repositories* was introduced. A Maven Repository is used to hold artifacts and dependencies of varying types. It can be either a local repository or a remote repository, accessible over a variety of protocols such as FTP or HTTP.

When needed the first time, a project's dependencies are fetched from such a remote repository. There are a lot of public repositories offering nearly all existing Java libraries with their describing POM files. Indirect dependencies (dependencies of dependencies) are also downloaded automatically if not prohibited in the project's setting.

The local repository is used for caching downloaded dependencies on the local machine in order to speed up further build processes.

[9] `http://ant.apache.org/`
[10] `http://maven.apache.org/`

By providing this well thought-out dependency management system, Maven makes organizing large projects very easy. For instance, it is possible to control the versions of all dependencies at one single point, a feature that significantly reduces the problem of version conflicts between different libraries.

Another mentionable advantage of Maven is its extensibility. There are a lot of plugins, allowing to integrate arbitrary actions into the build process. For instance, databases can be created, application servers can be started, the created application can be deployed and a lot of things more. Furthermore, it is possible to create project files for the Eclipse IDE and for other development environments, too.

Because of this unique features and because of the fact that modularity was an important design goal of Cloud42 (see Section 4.2.1) , Maven was preferred over Ant or other alternatives and Cloud42 was set up as a Maven-powered project.

4.4 User Interface

The Cloud42 web application provides a so-called "rich" GUI, which simply is a desktop-like user interface delivered over the Web and viewed in a browser, allowing the user to interact with Cloud42 in a convenient way.

This section provides a very short description of this GUI and highlights some technical backgrounds.

In order to use the features of the web application, an user has to create an account first, consisting of an username and a password for protection. After logging in for the first time, the AWS credentials can be entered. These are stored in the database for further usage, coupled to the user's account.

4.4.1 Main Screen

Figure 4.5 shows the main screen of Cloud42 that appears right after login (assuming credentials are already known).

As you can see, this screen provides the possibility to search for available AMIs or to register new ones. Of course, launching instances of these AMIs is possible, too. Furthermore, the second part of the screen contains a list of currently running instances with their properties.

All tables are sortable. Moreover, the upper table has filtering capabilities allowing to search for particular AMIs. For instance, in Figure 4.5 a filter is applied showing only AMIs owned by a user whose user-id starts with "59".

Basically, all interactions are accomplished using modal windows like it is used to be done in desktop applications. For instance, if the user clicks on one of the icons in the last cell of the lower table (the meanings of the icons are outlined by tooltips), a corresponding modal dialog panel appears. Figure 4.6 shows an example.

Each of the extended functionalities of Cloud42 can be accessed over an own modal dialog.

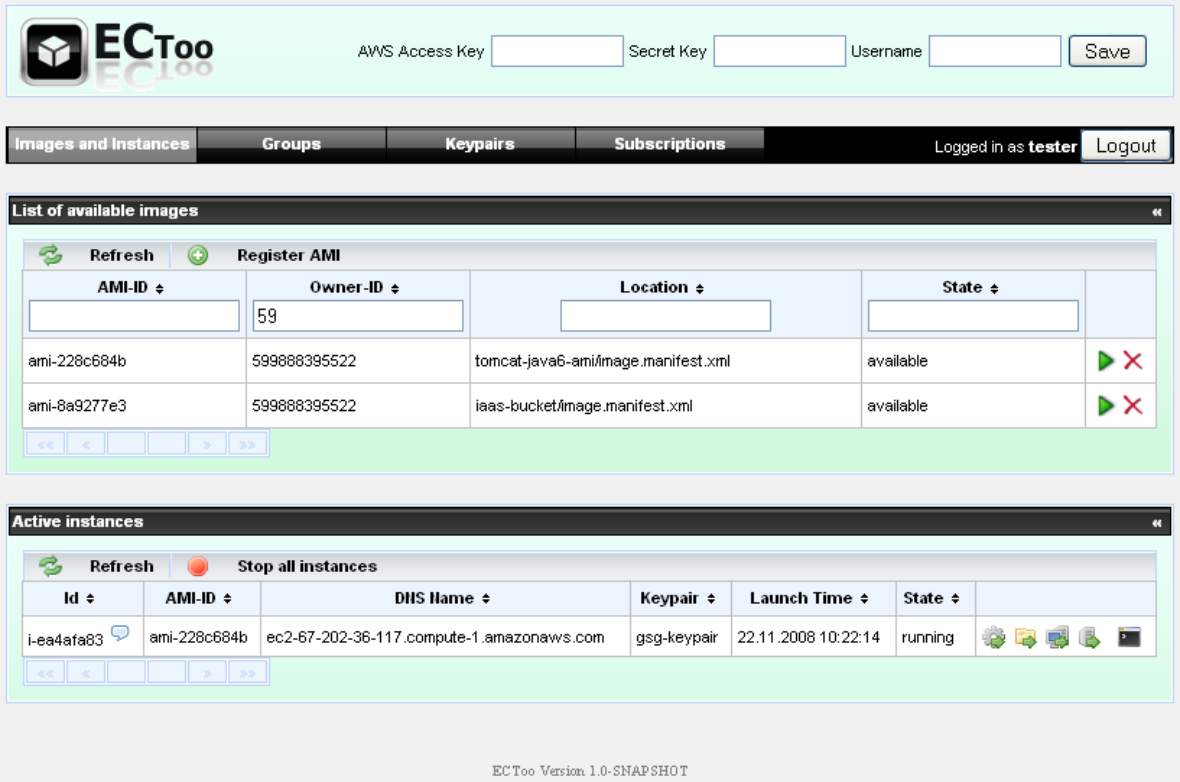

Figure 4.5: Cloud42 main screen

4.4.2 Security Groups Screen

The second screen of Cloud42 is the screen for editing EC2's security groups. Groups and their correnspondending rights are presented in a hierarchical view as you can see in Figure 4.7.

Editing groups is straight forward by selecting the desired action from a context menu that appears when double-clicking on a node. Unfortunately, it was not possible to react on right-click events due to technical constraints and browser incompatibilities in this case.

4.4.3 Keypairs Screen

Figure 4.8 shows the third screen of the Cloud42 web application.

Again, it is a quite simple screen that is used to create, edit and delete EC2 keypairs. In addition to creating and deleting keypairs, a user can assign a RSA private key to a keypair. The private key then is stored in the database of Cloud42. Assigning private keys is required in order to be able to execute the extended functionalities like file transfer that need to know the user's authentification information.

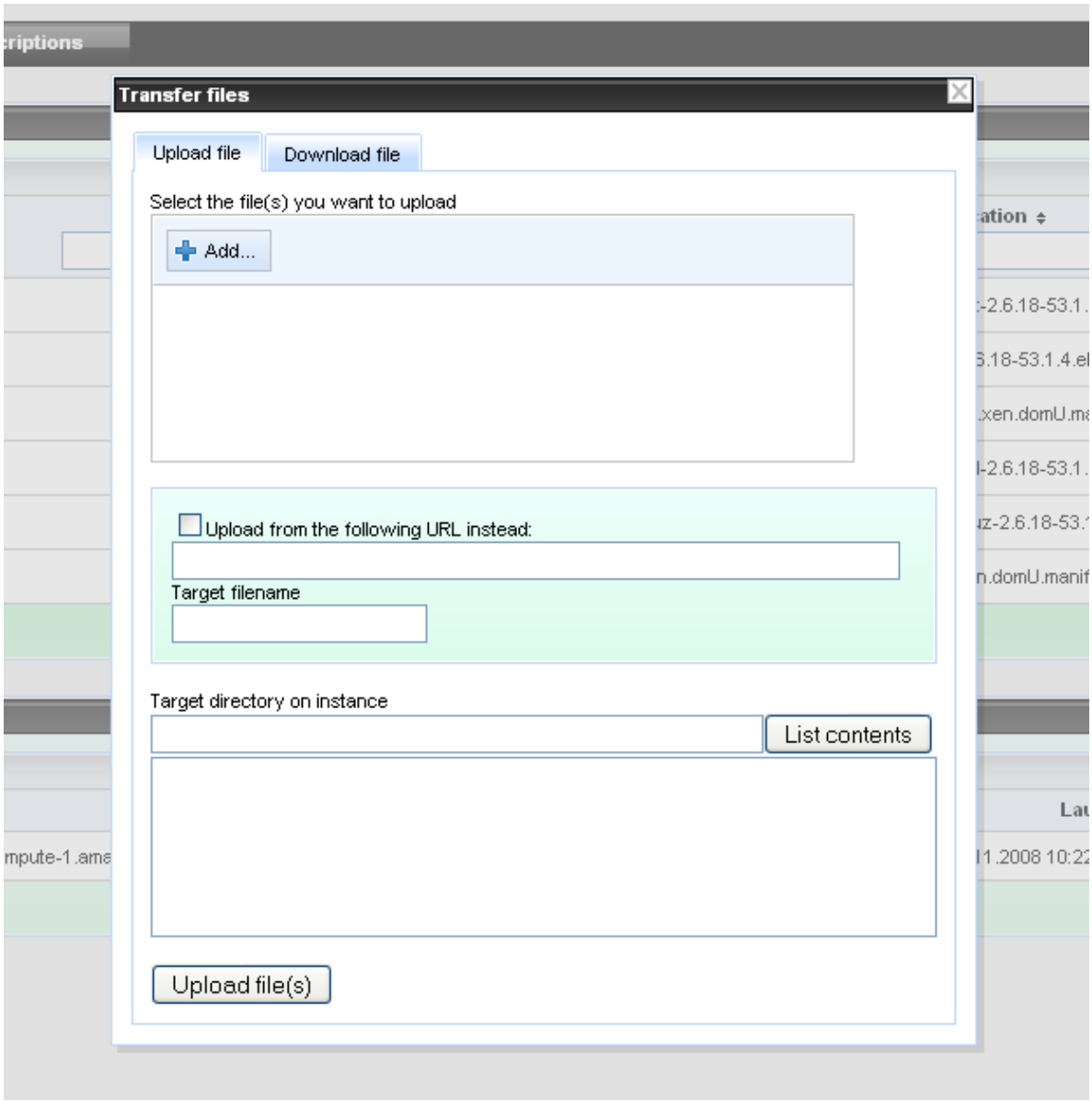

Figure 4.6: Cloud42 modal panel for file transfer

4.4.4 Subscriptions Screen

The forth and last screen serves for managing subscriptions for the Cloud42 notification mechanism elaborated in Section 3.4. Figure 4.9 presents a screenshot.

Usually, the Web service layer of Cloud42 is used to handle subscriptions. However, for convenience reasons, the user can list, subscribe and unsubscribe endpoints using the web application, too.

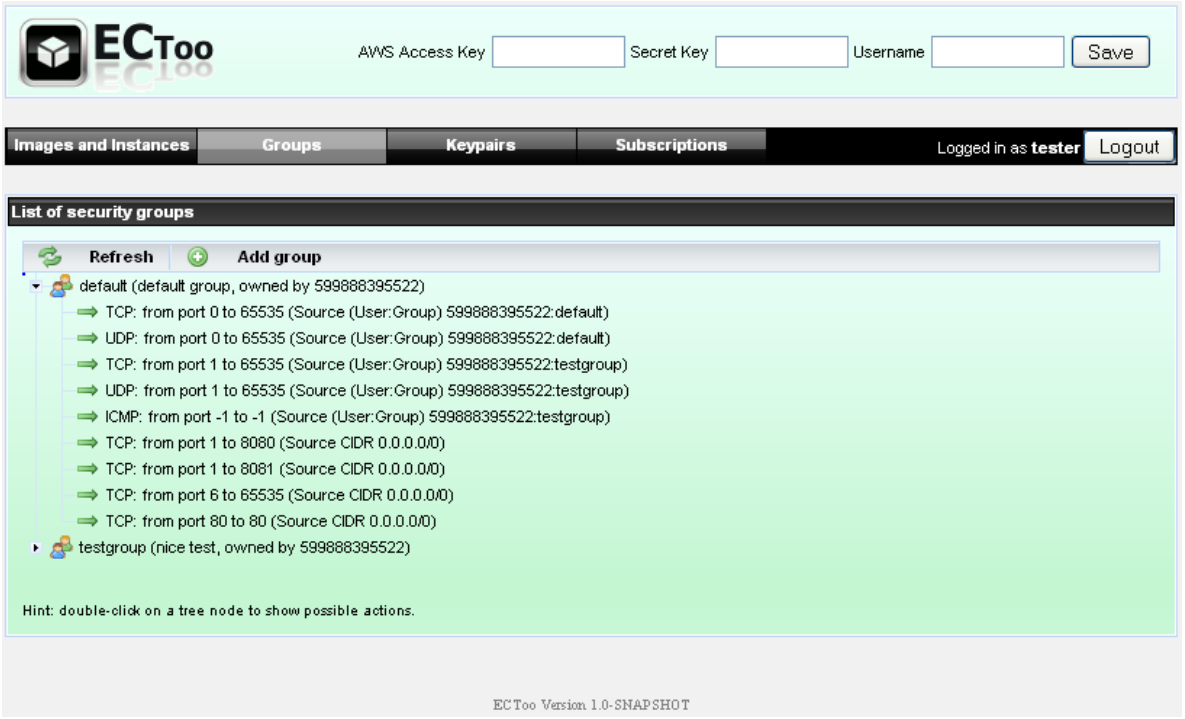

Figure 4.7: Cloud42 security groups screen

But there is a constraint: while the Web service layer is extensible and allows to subscribe any kind of endpoint, the GUI only supports subscribing endpoints that receive their notifications as SOAP messages (so-called `SOAPSubscriptions`, see Section 3.4.3 for details).

4.4.5 Technology

As mentioned above, the user interface of Cloud42 provides a desktop-like Look&Feel. This behaviour is enabled by the heavy use of AJAX functionalities throughout the web application.

Traditionally, a web application would consist of several views, each of them rendered as a different HTML page and having a different URL. This way, each interaction would cause the browser to send a new GET or POST request to the server and to completely load the resulting view.

For highly interactive websites, where the user either has to provide data frequently or data on the server often changes (or both, of course), this is an inacceptable drawback, since (re)loading complete views each time results in poor response times, forcing the user to wait instead of being able to continue work. As a conclusion. the application becomes unusable.

AJAX aims to overcome this drawback by allowing partial page rendering. Using a special *XMLHTTPRequest* and JavaScript, only relevant data is submitted to the server and the DOM Tree of the page is modified according to the response, so that it is not necessary to refresh to whole page itself.

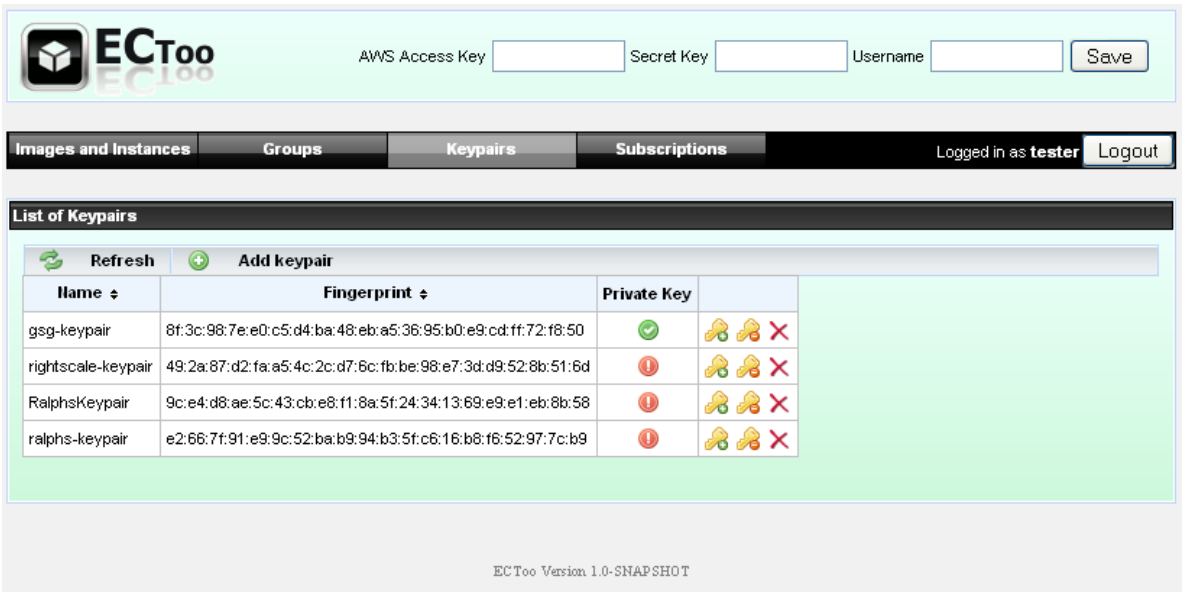

Figure 4.8: Cloud42 keypairs screen

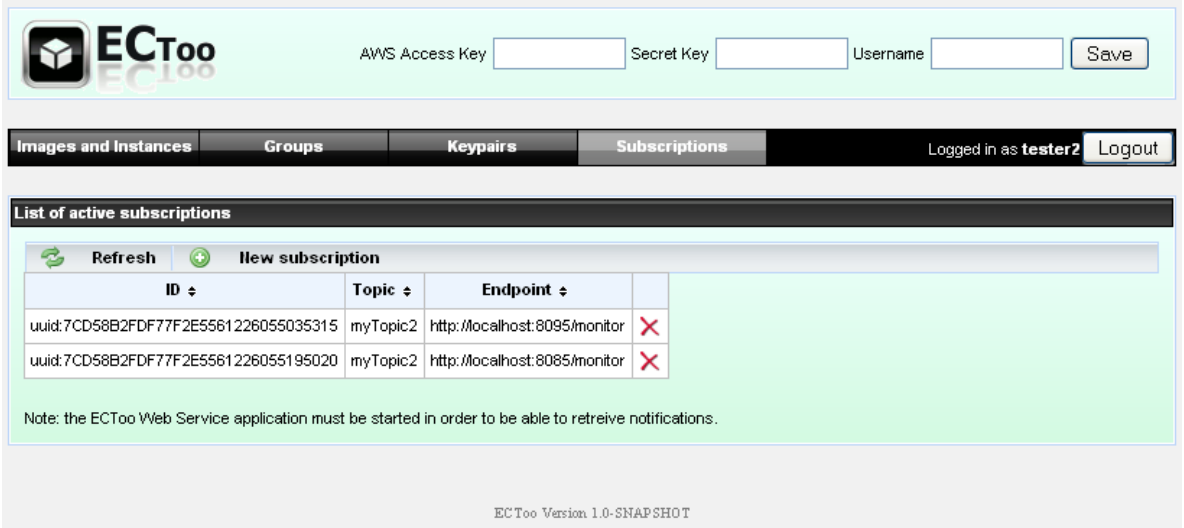

Figure 4.9: Cloud42 subscriptions screen

As already described in Section 4.3.4, the JBoss RichFaces component library for Java Server Faces is an ideal choice to create web applications supporting AJAX capabilities, since they provide AJAX features out of the box.

The Cloud42 web application uses these AJAX capabilities within each of the four screens.

For instance, submitting data from the file transfer dialog presented in Figure 4.6 does not force a complete reload of the page. Instead, only the corresponding form is submitted using an AJAX request.

When the result arrives from the server, the relevant parts of the page are updated. In this case, this includes resetting the input fields (so that another file can be uploaded quickly) and displaying a success or failure message.

Similar behavior applies to all the other modal dialogs of each screen.

Another good example are the lists contained on each screen (list of images, instances, groups, keywords, subscriptions). When the user hits the "Refresh" button for a particular list, only this specific list is reloaded from the server.

In the case of the main screen, this is an especially good improvement, since reloading the list of images and the list of instances requires requests to the EC2 API and therefore can be quite time-consuming. Refreshing the whole page would imply the reload of both lists which would cost twice the time of simply loading the desired list using an AJAX request.

4.5 Web Service Interface

As mentioned several times throughout this work, the Web service interface is the actual core of Cloud42, since it allows to invoke Cloud42's enhanced functionalities from within other software or BPEL processes very easily. This way, Cloud42 can be used to compose large software systems, exploiting all the benefits of Cloud Computing.

4.5.1 Design

The main challenge when designing the Cloud42 Web service layer was state handling. In order to interact with EC2, several user information like the AWS credentials (used to identify an user at the EC2 API) or RSA private keys have to be provided.

This raises the question whether such information should be stored within Cloud42 or whether a calling process should provide it within each request.

Whereas the Cloud42 web application clearly has to cache data for convenience reasons (otherwise the user had to enter his credentials all the time), the answer to this question is more difficult when looking at the Web service interface.

Since the Cloud42 user interface and the Cloud42 Web service should be able to run independently from each other, it is not possible to use the information stored in the database of the web application from within the Web service layer. Therefore, implementing an own state management and storage mechanism would be necessary.

Furthermore, stateful Web service interfaces are more difficult to handle in general, because the caller must be identified and some kind of session information must be transferred.

As a conclusion, the Cloud42 Web service interface was supposed to be fully stateless. Finally, this means the caller has to handle access information for itself. AWS credentials and other data are passed to the Cloud42 API within each actual request.

At a first glance, this does not seem to be an ideal solution. However, it guarantees full flexibility and scalability by delegating state management and responsibility to the calling process, allowing Cloud42 to focus on its main capabilities.

There is a last argument to highlight: in view of security considerations, transferring critical data like RSA private keys is problematic in few cases when using unreliable connections, even more if it occurs in each single request. This fact could be overcome by implementing WS-Security mechanisms ([Oas06]). However, due to compatibility reasons, Cloud42 does not support WS-Security for now, since this would require the client to use only tools implementing WS-Security and therefore is a serious constraint.

4.5.2 Services

The Cloud42 Web service interface is logically separated into four particular services, each of them providing its own WSDL file for full modularity.

- `Cloud42BaseService` contains the base methods to interact with EC2. Querying instance information, launching new instances or creating security groups are some examples.
- `Cloud42FileService` provides methods for up- and downloading files using MTOM according to section 3.2.
- `Cloud42NotificationService` is the implementation of the Cloud42 notification mechanism elaborated in section 3.4. It consists of the subscribe and unsubscribe methods. Furthermore, it allows configuring the address of the endpoint published by Cloud42 in order to retrieve event messages from running AMI instances.
- `Cloud42RemotingService` finally can be used to execute commands or batch files remotely on an EC2 server instance. Bundling new AMIs is also supported by this service.

4.5.3 BPEL Example

The following section illustrates an example BPEL process using the Cloud42 Web service API, since invoking EC2's functionalities from within BPEL was one of the main motivations behind this work.

A small and simple scenario was chosen: suppose a particular AMI is needed in order to do some calculations. However, it is not sure whether an instance of this AMI is already running or whether it is required to start a new one. The BPEL process presented in Figure 4.10 and Figure 4.11 can be used to automatically do this check.

The process requires AWS credentials (user id, key, secret key), a keypair to use and an image location (on Amazon S3) as input.

At first, it retrieves the EC2 `imageId` for the given AMI location by querying `listImages`.

Then, it checks whether an instance of this AMI is already running.

If so, it returns the EC2 `instanceId` of the first instance of this AMI.

Otherwise an instance of the given AMI is started by invoking the `runInstance` function. In this case, the `instanceId` of the newly created instance is returned to the client.

Although the sample process does only invoke functions from the `Cloud42BaseService` part of the Cloud42 Web service interface, it shows the power of this approach. More complex BPEL processes can be developed using the enhanced features of Cloud42 like uploading files or even executing arbitrary commands on AMI instances. This way, orchestrating EC2 becomes easy and convenient.

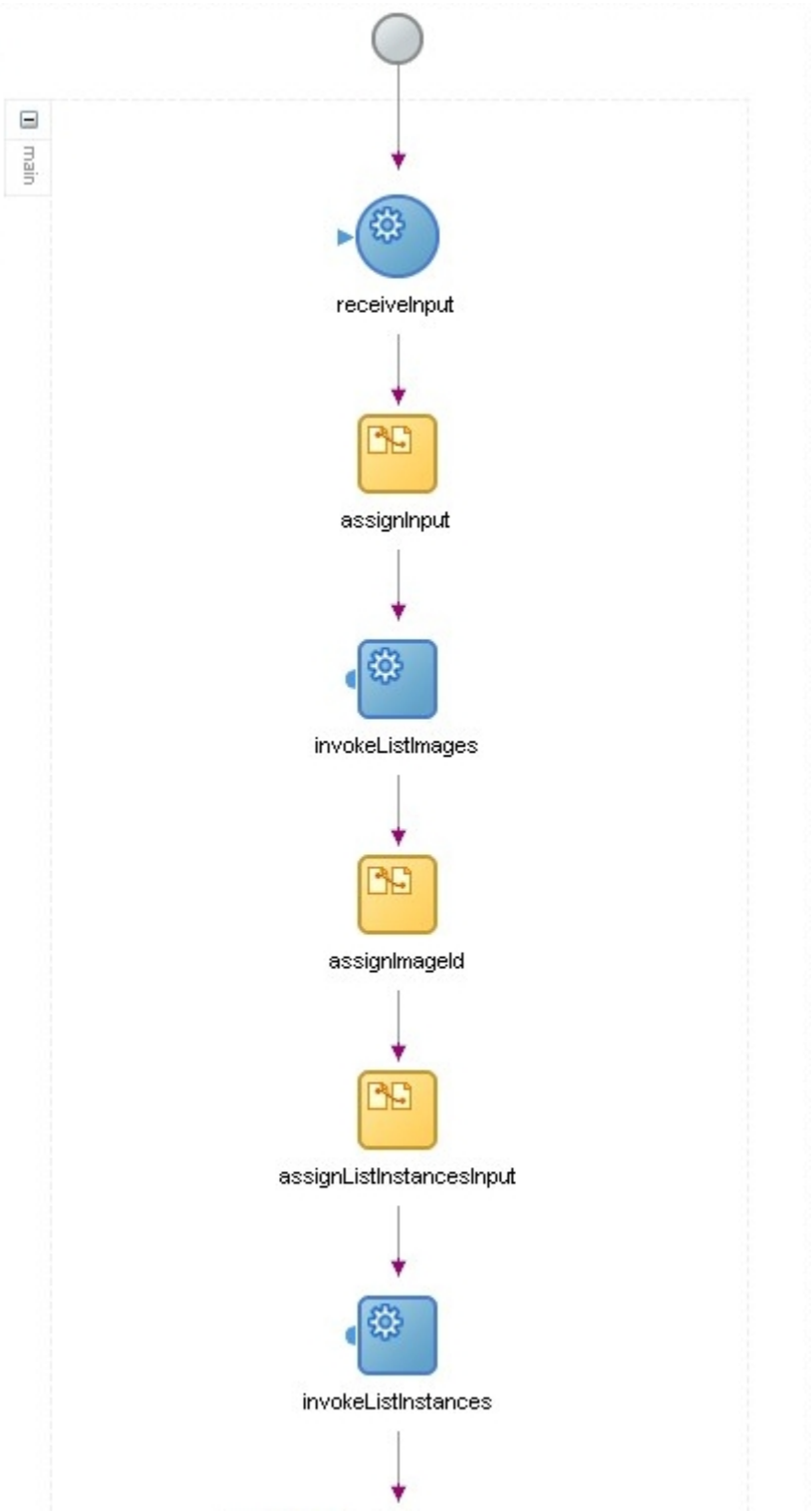

Figure 4.10: Sample BPEL process - Part 1

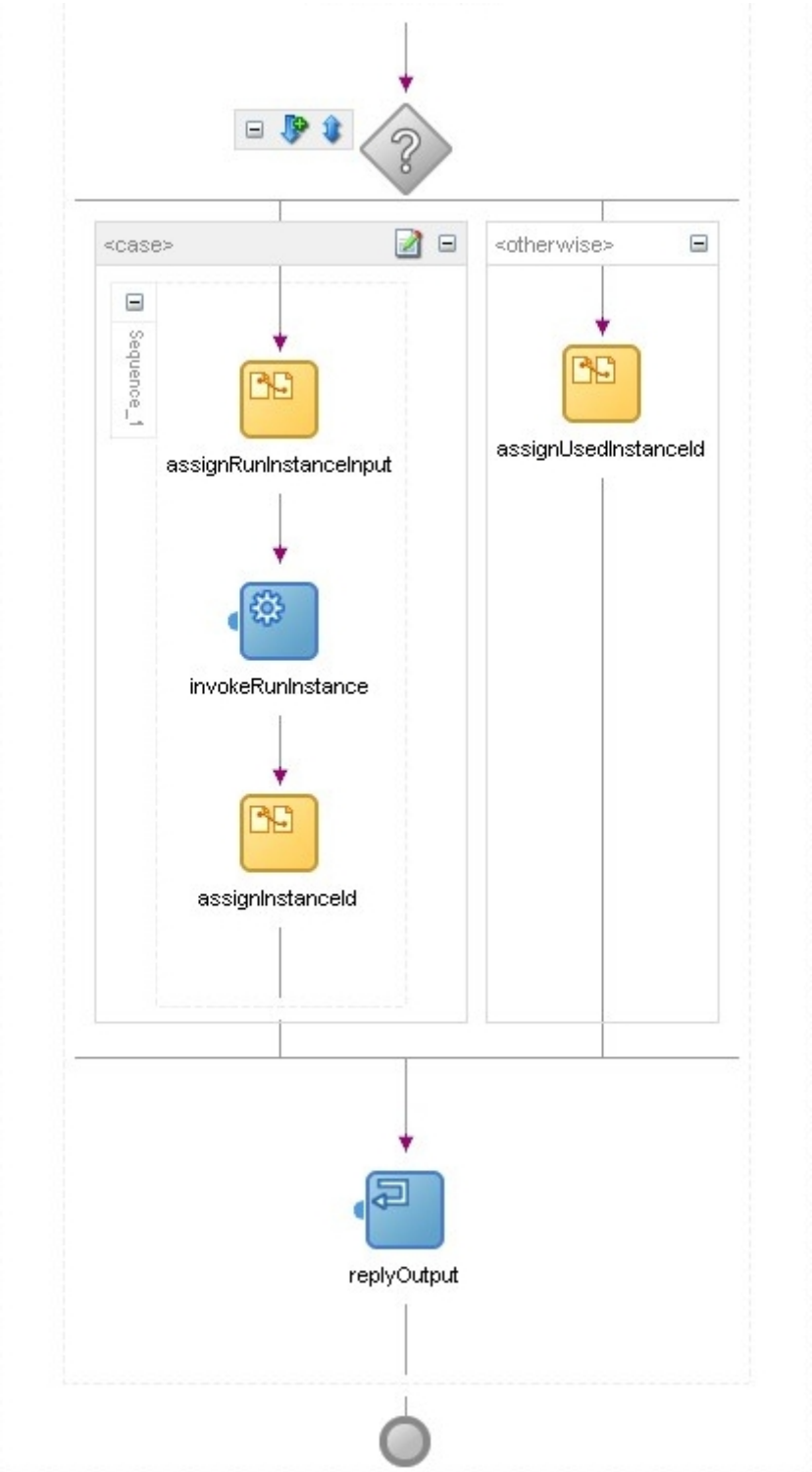

Figure 4.11: Sample BPEL process - Part 2

Chapter 5

Summary and Outlook

Cloud Computing in general and Infrastructure as a Service in special are very current issues in these days.

Being in a permanent change, a lot of research is done in this area. Current services are continuously enhanced. For instance, Amazon recently introduced a concept called *Elastic IP* which significantly increases the value of the Amazon Web Services by combining the advantages of static IP addresses with the dynamic of Cloud Computing. Several other features were added while this work was written, amongst them the probably most important one is the *Amazon Elastic Block Store*[1], a persistent storage for Amazon EC2 instances.

Furthermore, new service providers appear on the market. Microsoft announced a "cloud services operating system" called "*Windows Azure*"[2] on October 27, 2008.

According to a keynote speech on the Microsoft PDC '08[3] from Ray Ozzie, Microsoft's chief software architect, this platform will be a watershed for Microsoft, forming the strategy for the next ten years, a fact that truely outlines the relevance and the potential of Cloud Computing.

Although Windows Azure differs from EC2, because it tries to provide different services within one product and therefore misses some of the modularity and flexibility of EC2, it becomes clear that it is Microsoft's answer to the Amazon Web Services and an effort to being competitive with Google and Amazon in the future. As a first and very quick response, Amazon enhanced its range of products and presented "Amazon EC2 running Microsoft Windows Server"[4], a family of instance types that are based on the Windows Server 2003 operating system.

As mentioned several times, there are many scenarios where Cloud Computing enables interesting concepts or even new business models. Of course, the applications illustrated for instance in Section 2.4 do not require any specific service provider and can be realized using Amazon EC2 as well as Google's or Microsoft's competing services.

[1] `http://aws.amazon.com/ebs/`
[2] `http://www.microsoft.com/azure/windowsazure.mspx`
[3] `http://channel9.msdn.com/pdc2008/KYN01/`
[4] `http://aws.amazon.com/windows/`

However, the main part of this thesis concentrated on Amazon's Web Services. A definition of EC2 and the related services was given. Some related work was presented, outlining the particular intent of this thesis amongst others.

The final goal of this work was to develop a comprehensive management framework for EC2 with special focus on a clean Web service interface. The presence of such an interface allows managing and orchestrating EC2 server instances not only by human users, but also by other software or BPEL processes. This way, an abstraction from EC2 as a particular service provider can be reached.

Chapter 3 introduced some enhanced concepts that represent valuable functionalities for a management tool for EC2. The base requirement in order to be able to interact with an AMI instance is the possibility to execute arbitrary commands on it. This way, an instance can be configured and accessed remotely, using the Web service interface of the tool. Furthermore, this forms the basis for more complex tasks like uploading files or bundling new AMIs.

Finally, an extensible notification mechanism was elaborated, allowing to subscribe arbitrary endpoints to event messages from AMI instances using a publish/subscribe pattern.

All of these concepts were put into practice in the implementational part of this work. The actual implementation of the management tool with the name "Cloud42" was described in Chapter 4. A brief overview of the general architecture was given and the technologies and frameworks used for implementation were introduced. Some of the main features both of the user interface and the Web service interface were highlighted.

With its wide range of functions, Cloud42 can serve as a web-based, graphical management tool for human users as well as to provide a Web service API for other applications and therefore fullfills the goal of this work.

However, there are still some possible enhancements: due to the fact that Amazon updates and enhances its API frequently, not all of the new features are usable from within Cloud42. For instance, support for Elastic IP adresses is missing. Because of this, the next step would be to adopt Cloud42 to the newest version of the Amazon EC2 API.

Intended to be published under an Open Source licence, Cloud42 will experience ongoing development in future in any case. The project website is available at `http://cloud42.net`.

As a second step, applications based on Cloud42 could be developed, using the Web service API to orchestrate AMI instances and to manage whole clouds. Cloud42 could be used as a provisioning service for EC2 within a more abstract provisioning engine as proposed by [ML08], empowering the usage of Amazon in combination with other service providers.

Appendix A

Sample Implementation for SMS Notification

In order to show the extensibility of the Cloud42 notification mechanism, the following sections demonstrate how to implement a SubscriptionProcessor that can handle sending notification messages via SMS. It is recommended that you read section 3.4 first for a better understanding. Since the referenced section does not cover implementational details, having a look at the existing code is also a good idea to get a feeling which interfaces are going to be implemented.

Mainly, empowering the notification mechanism to support a new kind of endpoint consists of three steps.

A.1 Defining a Subscription Message

The first step is to define a subscription message that is going to be parsed by the new SubscriptionProcessor.
In the case of a SMS notification, the subscribing endpoint is just a telephone number, so a subscription request could look like the one presented in Listing A.1

As `<web:subscriptionProcessor>` we have to specify the class that will finally process the subscription request message. In this case, it is the `my.company.SMSSubscriptionProcessor` that will be implemented in the next step.

The variable part of the message included in the `<web:subscriptionMessage>` element contains a telephone number representing the subscribing endpoint.

Listing A.1 Example for a SMS subscription request message

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:web="http://webservice.cloud42.jw.de">
  <soap:Header/>
  <soap:Body>
    <web:subscribe>
      <web:subscriptionProcessor>
        my.company.SMSSubscriptionProcessor
      </web:subscriptionProcessor>
      <web:topic>myTopic</web:topic>
      <web:subscriptionMessage>
        <tb:Entry xmlns:tb="http://my.company.com/telephonebook">
          <tb:Number>
            408 726 4390
          </tb:Number>
        </tb:Entry>
      </web:subscriptionMessage>
    </web:subscribe>
  </soap:Body>
</soap:Envelope>
```

A.2 Implementing a SubscriptionProcessor

Now we have to implement the `SMSSubscriptionProcessor` class. It will parse the subscription message and create a corresponding Subscribtion object, namely a `SMSSubscription` (see next section).

Cloud42 provides a generic SubscriptionProcessor called `GenericSubscriptionProcessor` that all other SubscriptionProcessors should inherit from, since it offers some generally usable methods, mainly for processing unsubscription requests and handling subscription ids.

So the `my.company.SMSSubscriptionProcessor` looks like shown in Listing A.2.

A.3 Implementing the Subscription Class

As a last step, we have to implement the `SMSSubscription` class containing the algorithms to convert and send the notification messages as SMS. For this purpose, Cloud42 offers an abstract superclass called `Subscription` that determines the methods a Subscription class must implement. Furthermore, it provides methods for getting and setting both the subscription Id and the topic. Listing A.3 shows how an actual implementation could look like.

Finally, Hibernate requires a mapping for the newly created class in order to be able to persist it in the database. Thanks to using Hibernate Annotations, this can be done by simply adding the line presented in Listing A.4 to the file `hibernate.cfg.xml` that is contained in `module-hibernate`.

That is all that has to be done. There is no need to change any existing code or to adjust any further configuration file. By sending request messages like the one in Section A.1, a subscriber now can register any mobile phone number for retrieving notifications on the specified topic via SMS.

Listing A.2 Implementation of `my.company.SMSSubscriptionProcessor`

```
package my.company;

//imports are missing due to readability

public class SMSSubscriptionProcessor extends GenericSubscriptionProcessor {

	/**
	 * This method parses the subscriptionMessage element, extracts the subscribing
	     endpoint (a telephone number) and returns a SMSSubscription object.
	 *
	 */
	@Override
	public Subscription getSubscriberFromMessage(OMElement message) throws
	    EventingException{

		//create SMSSubscription instance
		SMSSubscription subscription = new SMSSubscription();

		//parse message, extract telephone number
		OMElement entryElement = message.getFirstChildWithName(new QName(
				Constants.TELEPHONEBOOK_NAMESPACE,
				Constants.ElementNames.Entry));
		if (entryElement == null)
			throw new EventingException("Entry element is not present");

		OMElement numberElement = entryElement
				.getFirstChildWithName(new QName(
						Constants.TELEPHONEBOOK_NAMESPACE,
						Constants.ElementNames.Number));
		if (numberElement == null)
			throw new EventingException("Number element is not present");

		String number = numberElement.getText();


		//set the telephone number
		subscription.setTelephoneNumber(number);


		return subscription;

	}

}
```

Listing A.3 Implementation of `my.company.SMSSubscription`

```
package my.company;
//imports are missing due to readability
@Entity //mark as entity in order to be able to save SMSSubscriptons in the datatbase
public class SMSSubscription extends Subscription {
	/**
	 * Telephone number where notifications must be sent to.
	 */
	private String number;
	/**
	 * @return the telephone number for this subscription
	 */
	public String getTelephoneNumber() {
		return number;
	}
	/**
	 * @param number the telephone number to set for this subscription
	 */
	public void setTelephoneNumber(String number) {
		this.number = number;
	}
	/**
	 * Transforms the provided notification message into a SMS and sends it.
	 *
	 * @param message internal representation of the message to send.
	 */
	@Override
	public void sendEventData(Message message) throws Exception {
		//send a SMS to the subscribed telephone number

		//create human readable SMS text with message information
		String text = "Notification message on topic " + message.topic
				+ " from AMI instance " + message.instanceId
				+ ". Time: " + message.timestamp + ". Event description is: "
				+ message.text + ". Additional information: " + message.info;

		//now send this SMS using a fictive SMS library
		MySMSLibrary.sendSMS(number, text);
	}
}
```

Listing A.4 Hibernate Mapping for `my.company.SMSSubscription`

```
<mapping class="my.company.SMSSubscription"/>
```

Abbreviations

AJAX	Asynchronous JavaScript and XML
AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
BPEL	Business Process Execution Language
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
JSF	Java Server Faces
JSP	Java Server Pages
MVC	Model View Controller
ORM	Object-relational Mapping
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SSH	Secure Shell
URL	Uniform Resource Locator
VNC	Virtual Network Computing
WS	Web Service
WSDL	Web Services Description Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Bibliography

[AW08] B. Aranda, Z. Wadia. *Facelets Essentials*. Apress, 2008. (Cited on page 50)

[Bar08] D. Baran. Cloud Computing Basics. 2008. URL `http://www.webguild.org/2008/07/cloud-computing-basics.php`. (Cited on page 15)

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*. Wiley, 1996. (Cited on page 47)

[Cab07] P. Cabrera. Using Parameterized Launches to Customize Your AMIs. 2007. URL `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1085`. (Cited on page 31)

[Cab08] P. Cabrera. Patching AMI Instances with Updates on Amazon S3. 2008. URL `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1403`. (Cited on page 31)

[Cho07] I. Choudhary. Software as a Service: Implications for Investment in Software Development. *hicss*, 00:209a, 2007. doi:http://doi.ieeecomputersociety.org/10.1109/HICSS.2007.493. (Cited on page 15)

[Fie00] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. 2000. URL `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`. (Cited on page 39)

[Fou08] T. A. S. Foundation. *Axis2 Architecture Guide*. 2008. URL `http://ws.apache.org/axis2/1_4_1/Axis2ArchitectureGuide.html`. (Cited on page 53)

[FR08] J. S. Forrester Research, Inc. Is Cloud Computing Ready For The Enterprise? 2008. (Cited on page 13)

[Fro08] J. Fronckowiak. Auto-Scaling Web Sites Using Amazon EC2 and Scalr. 2008. URL `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1603`. (Cited on page 23)

[Gar07] S. L. Garfinkel. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical Report TR-08-07, Harvard University, 2007. URL `http://www.simson.net/clips/academic/2007.Harvard.S3.pdf`. (Cited on page 11)

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. (Cited on page 47)

[GNC+] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl. Publish-Subscribe Notification for Web services. URL `http://www.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf`. (Cited on page 35)

[Jon07] A. de Jonge. Deploying Distributed J2EE Applications Using Amazon EC2. 2007. URL `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1084&categoryID=100`. (Cited on page 11)

[Jos05] A. P. Joshi. Design with the JSF architecture. 2005. URL `http://www.ibm.com/developerworks/web/library/wa-dsgnpatjsf.html`. (Cited on page 48)

[KNL08] T. Kwok, T. Nguyen, L. Lam. A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application. *scc*, 2:179–186, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/SCC.2008.138. (Cited on page 15)

[Lan08a] K. Langley. Cloud Computing: Get Your Head in the Clouds. 2008. URL `http://www.productionscale.com/home/2008/4/24/cloud-computing-get-your-head-in-the-clouds.html`. (Cited on page 15)

[Lan08b] K. Langley. Is SaaS Cloud Computing? 2008. URL `http://www.productionscale.com/home/2008/6/29/is-saas-cloud-computing.html`. (Cited on page 15)

[Law08] G. Lawton. Developing Software Online With Platform-as-a-Service Technology. *Computer*, 41(6):13–15, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2008.185. (Cited on page 16)

[LLC08] A. W. S. LLC. Instance Metadata - Amazon Elastic Compute Cloud Developer Guide. 2008. URL `http://docs.amazonwebservices.com/AWSEC2/2008-02-01/DeveloperGuide/AESDG-chapter-instancedata.html`. (Cited on pages 31 and 39)

[LO08] H. Liu, D. Orban. GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications. *ccgrid*, 0:295–305, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/CCGRID.2008.30. (Cited on page 11)

[Mil08] M. Miller. *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. Que Corp, 2008. (Cited on page 11)

[ML08] R. Mietzner, F. Leymann. Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. *services*, 0:3–10, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/SERVICES-1.2008.36. (Cited on pages 9, 11 and 66)

[Mur08] J. Murty. *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Media, Inc., 2008. (Cited on page 11)

[Nic08] M. Nicholls. Amazon EC2 and what it means for Entrepreneurs and Startups. 2008. URL http://searchcorp.biz/amazon-ec2-and-what-it-means-for-entrepreneurs-and-startups. (Cited on page 24)

[OAS] OASIS. Web Services Notification. URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn. (Cited on page 35)

[Oas06] Oasis. Web Services Security: SOAP Message Security 1.1. 2006. URL http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. (Cited on page 61)

[OC08] S.-P. Oriyano, P. Cabrera. Introduction to Software Load Balancing with Amazon EC2. 2008. URL http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1639. (Cited on page 23)

[Var08] J. Varia. Building GrepTheWeb in the Cloud, Part 1: Cloud Architectures. 2008. URL http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1632. [Online; accessed 7-August-2008]. (Cited on page 8)

[W3C00] W3C. SOAP Messages with Attachments, W3C Note 11 December 2000. 2000. URL http://www.w3.org/TR/SOAP-attachments. (Cited on page 29)

[W3C05a] W3C. SOAP Message Transmission Optimization Mechanism, W3C Recommendation 25 January 2005. 2005. URL http://www.w3.org/TR/soap12-mtom/. (Cited on page 29)

[W3C05b] W3C. XML-binary Optimized Packaging, W3C Recommendation 25 January 2005. 2005. URL http://www.w3.org/TR/xop10/. (Cited on page 29)

[W3C06] W3C. Web Services Eventing (WS-Eventing), W3C Member Submission 15 March 2006. 2006. URL http://www.w3.org/Submission/WS-Eventing/. (Cited on pages 35 and 37)

[W3C07] W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007. 2007. URL http://www.w3.org/TR/soap12-part1/. (Cited on page 53)

Unless otherwise noted, all links were last followed on November 24, 2008.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Frank Bitzer)